

DOCUMENT RESUME

ED 070 504

LI 004 059

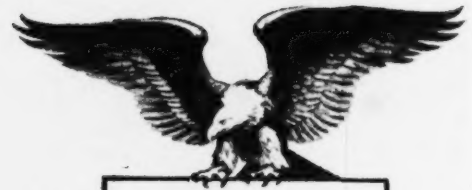
AUTHOR Glantz, Richard S.  
TITLE Design of an On-Line Query Language for Full Text  
Patent Search.  
INSTITUTION Mitre Corp., McLean, Va.  
SPONS AGENCY Patent Office (DOC), Washington, D.C.  
REPORT NO USPO-M72-97  
PUB DATE 28 Sep 72  
NOTE 82p.; (4 References)  
  
EDRS PRICE MF-\$0.65 HC-\$3.29  
DESCRIPTORS Computers; \*Information Retrieval; Information  
Storage; Information Systems; On Line Systems;  
\*Patents; \*Programing Languages; \*Search  
Strategies

ABSTRACT

The design of an English-like query language and an interactive computer environment for searching the full text of the U.S. patent collection are discussed. Special attention is paid to achieving a transparent user interface, to providing extremely broad search capabilities (including nested substitution classes, Kleene star events, and domain restrictions) and to specifying the query language by means of a formal grammar and an associated parsing mechanism which particularly facilitates error detection and reporting. A description of an initial working prototype system on the IBM 360/370 is also included. (Author)

M72-97

U.S. DEPARTMENT OF  
EDUCATION  
OFFICE OF EDUCATION  
THIS DOCUMENT IS  
DUCED EXACTLY AS  
THE PERSON OR  
INATING IT POINTS  
IONS STATED DO NOT  
REPRESENT OFFICIAL  
CATION POSITIONS



united  
states  
patent  
office

**DESIGN OF AN  
ON-LINE QUERY LANGUAGE  
FOR FULL TEXT PATENT SEARCH**

I 004 059

FILMED FROM BEST AVAILABLE COPY

MITRE  
CORPORATION

TYPOGRAPHICAL ERRORS

December 1972

in

"Design of an On-Line Query Language for Full Text Patent Search"

| Page-Par-Line | As Printed                                                  | Correct                                                      |
|---------------|-------------------------------------------------------------|--------------------------------------------------------------|
| iv-4-1        | three                                                       | four                                                         |
| 2-ftnot       | House of Representative                                     | House of Representatives                                     |
| 4-2-1         | quasi-English                                               | quasi-English                                                |
| 5-3-6         | brief excursion                                             | brief excursion                                              |
| 6-4-3         | user action                                                 | user action:                                                 |
| 6-4-6         | vis-a-vis                                                   | vis-a-vis                                                    |
| 8-1-8         | bold face                                                   | boldface                                                     |
| 8-2-3         | Patent sections,                                            | Patent sections                                              |
| 10-2-1        | search function,                                            | search function;                                             |
| 12-3-3        | "search, "result, "state.                                   | "search, "result, "state.                                    |
| 12-4-2        | \$fruit, \$stdsearch                                        | \$fruit, \$stdsearch                                         |
| 14-0-4        | "@@"                                                        | @@                                                           |
| 16-2          | CONCATENATION                                               | CONCATENATION                                                |
| serious error | with indefinite span                                        |                                                              |
| 16-2          | [insert after PHRASING]                                     | CONCATENATION fresh then plums fresh * plums                 |
| serious error |                                                             | with indefinite span                                         |
| 17-0          | 32 [both instances]                                         | '32'                                                         |
| 17-0          | 46 [both instances]                                         | '46'                                                         |
| 30-2-2        | find '\$pattern'                                            | find '\$pattern'                                             |
| 31-0-2        | '2' stands                                                  | 2 stands                                                     |
| 31-0-3        | '2' stands                                                  | 2 stands                                                     |
| 33-1-13       | find $\neg(\neg \text{pear} \ \& \ \text{plum})/\text{par}$ | find $\neg(\neg(\text{pear} \ \& \ \text{plum})/\text{par})$ |
| serious error |                                                             |                                                              |
| 37-2-4        | in the data area a recapitulation                           | a recapitulation in the data area                            |
| 45-1-2        | MARK                                                        | MARK<br>^                                                    |
| 52-1-5        | there                                                       | others                                                       |
| 57-3-4        | status<br>restore<br>: data<br>frame                        | status<br>restore<br>: data<br>frame                         |
| 65-2-3        | "list                                                       | "list                                                        |
| 65-2-5        | "list                                                       | "list                                                        |

ED 070504

M72-97

DESIGN OF AN ON-LINE QUERY LANGUAGE  
FOR FULL TEXT PATENT SEARCH

by

Richard S. Glantz

28 September 1972

Prepared for the

UNITED STATES PATENT OFFICE

Under Contract 1-35539

#### ABSTRACT

The design of an English-like query language and an interactive computer environment for searching the full text of the U.S. patent collection are discussed. Special attention is paid to achieving a transparent user interface, to providing extremely broad search capabilities (including nested substitution classes, Kleene star events, and domain restrictions) and to specifying the query language by means of a formal grammar and an associated parsing mechanism which particularly facilitates error detection and reporting. A description of an initial working prototype system on the IBM 360/370 is also included.

#### ACKNOWLEDGEMENT

The author wishes to acknowledge the influence of the following people on the design of the query language and its interactive environment:

*Henry A. Bayard  
Stanley F. Cohen  
Carl Engelman*

Their well-reasoned suggestions and their feigned hands-in-the-air horror at some of my inanities have influenced me to rework parts of the design.

Thanks are also due to those who had a hand in designing, programming, and validating the prototype system:

|                            |                           |
|----------------------------|---------------------------|
| <i>Sheila M. Barry</i>     | <i>Samuel M. Gervasi</i>  |
| <i>Henry A. Bayard</i>     | <i>Louis N. Gross</i>     |
| <i>Judith A. Clapp</i>     | <i>Elyce M. Haut</i>      |
| <i>Robert J. Fleischer</i> | <i>Verniece Hensey</i>    |
| <i>Jeanne Fleming</i>      | <i>Joseph E. Sullivan</i> |
| <i>W. Reid Gerhart</i>     |                           |

A special debt is owed to Mr. Bayard, for his dogged perserverance in assembling together into a running system the "compatible" vendor-supplied software and hardware components; and to Mr. Sullivan, for sharing his extraordinary knowledge of PL/I which seemed to be missing from the manuals.

Of course, the project never would have been completed within the budget and on schedule without the watchful eyes of:

*Richard H. Bullen, Jr.  
Judith A. Clapp  
Ruth G. McCarthy*

The three "golf balls" needed to type this report were wielded by *Margaret Hooper*.

## TABLE OF CONTENTS

|                                                         | <u>Page</u> |
|---------------------------------------------------------|-------------|
| LIST OF ILLUSTRATIONS                                   | vii         |
| CHAPTER A      THE PROBLEM                              | 1           |
| BRIEF SUMMARY                                           | 1           |
| TECHNOLOGICAL POSSIBILITIES                             | 2           |
| CHAPTER B      THE WORK STATEMENT                       | 4           |
| REQUIREMENTS OF THE RFP                                 | 4           |
| CHAPTER C      OVERVIEW OF CAPABILITIES                 | 5           |
| CHAPTER D      PATENT STRUCTURE                         | 8           |
| CHAPTER E      QUERY LANGUAGE STRUCTURE AND CONVENTIONS | 10          |
| THE BASIC COMMANDS                                      | 10          |
| MODES OF INTERACTION                                    | 10          |
| DIALECTS                                                | 11          |
| COMMAND STRUCTURE                                       | 11          |
| IMPLEMENTATION CONSIDERATIONS                           | 12          |
| CHAPTER F      SEARCHING                                | 15          |
| SYNTAX OF THE COMMAND                                   | 15          |
| TABLE OF SEARCH CAPABILITIES                            | 16          |
| EXAMPLES OF SEARCH PRESCRIPTIONS                        | 17          |
| OPERATOR PRECEDENCE                                     | 19          |
| FORMAL GRAMMAR OF THE SEARCH PRESCRIPTION               | 21          |
| INTERMEDIATE FORM OF THE ARGUMENTS                      | 27          |
| CONVENTIONS AND DEFAULTS                                | 30          |
| SUBTLETIES TO BE AWARE OF                               | 32          |
| TRANSLATION PROCEDURE                                   | 33          |
| NORMAL AND ERROR RESPONSES                              | 35          |
| NATURE OF THE SEARCH PRODUCT                            | 37          |
| CHAPTER G      DEFINING                                 | 39          |
| SYNTAX OF THE COMMAND                                   | 39          |
| EXAMPLES OF USE                                         | 39          |
| OTHER COMMANDS ASSOCIATED WITH DEFINING                 | 42          |
| NORMAL AND ERROR RESPONSES                              | 43          |
| NATURE OF THE DEFINITION PRODUCT                        | 43          |

# TABLE OF CONTENTS (CONC.)

|                                           | <u>Page</u> |
|-------------------------------------------|-------------|
| CHAPTER H      BROWSING                   | 44          |
| GENERAL CAPABILITIES                      | 44          |
| SYNTAX OF THE COMMAND                     | 45          |
| EXAMPLES OF USE -- SEARCH PRODUCT         | 46          |
| EXAMPLES OF USE -- DEFINITION PRODUCT     | 47          |
| MOVEMENT ALONG THE SCROLL                 | 50          |
| NORMAL AND ERROR RESPONSES                | 53          |
| NATURE OF THE DISPLAY PRODUCT             | 54          |
| CHAPTER I      MISCELLANEOUS CAPABILITIES | 55          |
| MULTIPLE COMMANDS                         | 55          |
| PROMPTING                                 | 55          |
| RESTORING THE DISPLAY SCREEN              | 57          |
| LIBRARIES                                 | 58          |
| INTERRUPT                                 | 58          |
| APPENDIX I      PATENT SECTION NAMES      | 59          |
| APPENDIX II     THE PROTOTYPE SYSTEM      | 61          |

# LIST OF ILLUSTRATIONS

| <u>Figure Number</u> |                                                                                                                                                                                                                                                                                                              | <u>Page</u> |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| 1                    | The United States Patent Office in Arlington Virginia                                                                                                                                                                                                                                                        | viii        |
| 2                    | The manual system: searching through patents in the public search room                                                                                                                                                                                                                                       | 3           |
| 3                    | The computer system: display terminal used to access patents stored in a remote computer.                                                                                                                                                                                                                    | 71          |
| 4                    | An early experimental form of the screen layout, exemplifying character quality and screen capacity. The division into the four areas of system status, user input, system response, and data has been retained; the query language and the system messages have been revised somewhat in the final version. | 72          |
| 5                    | Keyboard engraving                                                                                                                                                                                                                                                                                           | 73          |

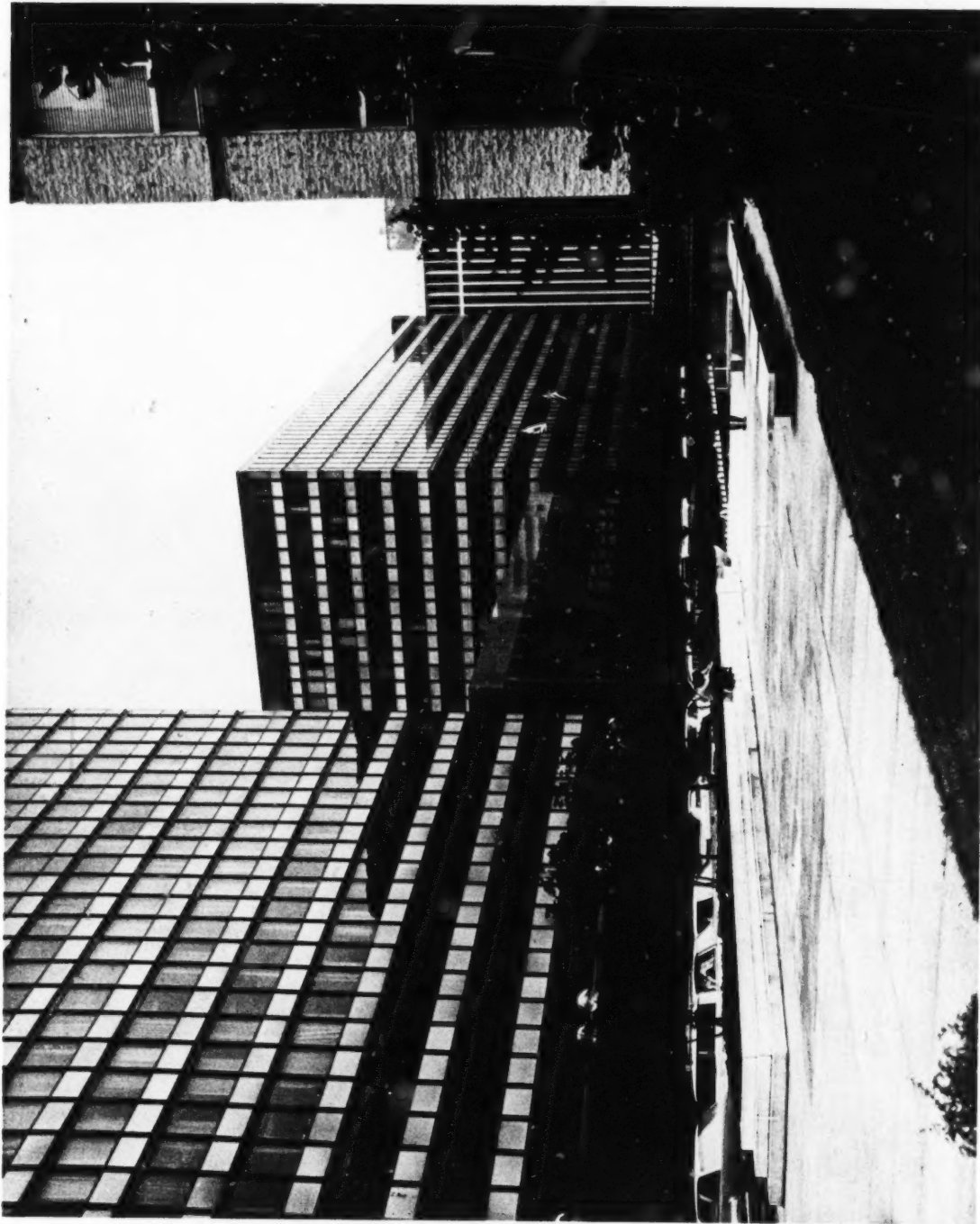


Figure 1. The United States Patent Office in Arlington Virginia  
(courtesy U.S. Patent Office)

## CHAPTER A

### THE PROBLEM

#### BRIEF SUMMARY

As a result of the rapid development of technology, the task of the Patent Office has become a prodigious one. Currently there are over 11 million patents inventoried, about 3.5 million of which are United States patents. The acquisition rate for this collection has spurted to 66,000 and above (U.S. patents) since 1966, whereas for the 15-year period prior to that, the average per annum rate was 45,000 (U.S.) patents. By contrast, the load on the examining staff, as measured by the number of patent applications filed each year, has shown a steady increase over the same 20-year time span from 60,000 per annum to a record high of 104,000 in fiscal year 1971.<sup>†</sup>

The primary task of the examiner is to determine the novelty of the ideas claimed by a patent applicant against the ideas already revealed in the open literature and the existing patent collection. Over this growing information store, the Patent Office has defined a classification schedule comprising 350 major classes and 80,000 subclasses. Because maintenance of the classification schedule has not kept pace with advancing technology, about 25% of the subclasses are unofficial; that is, they have not been formally defined and approved for general use within the Patent Office. In addition, changes in technology make it necessary to reclassify patents at a rate which is not commensurate with the amount of manual effort required. Consequently, the classes or subclasses to which these patents are assigned may no longer be appropriate.

In processing a patent application, the examiner relies on the classification to manually scan as many as 1,000 to 2,000 documents. From this set he extracts a subset which he feels contains material sufficiently relevant to the ideas claimed in the application to warrant taking several documents back to his office for closer inspection. This fine sieving typically produces about a dozen patents.

Current manual methods of conducting the examination of patent applications have resulted in an unreasonable delay in application processing. The average pendency is close to 29 months, and the

---

<sup>†</sup> FY 1972 figures were not available at the time this report was written.

backlog (date application filed until date examination begins) is almost a year. Both the Patent Office and the Congress<sup>†</sup> are insisting on a reduction of these figures.

This report addresses one aspect of the problem. It contains the design of a semi-automated system which should enable the patent examiner to retrieve relevant patents in a more efficient and effective manner while at the same time reducing the cost of maintaining the classification scheme.

#### TECHNOLOGICAL POSSIBILITIES

The same advances in technology which have made the task of the Patent Office a more burdensome one also offer the opportunity to alleviate the situation. Computer hardware technology has advanced to the point where several manufacturers now build memory storage units with capacities approaching billions of characters which can contain the vast collection of patent information and provide access to it at reasonable speed. Optical readers are available which can scan printed matter and convert it to a form acceptable as input to a computer. (Even optical readers won't be necessary for patents issued after July 1970: such patents are keyboarded onto magnetic tape to drive the Linotron at the Government Printing Office, and therefore the tapes containing the full patent text are already available in machine readable form.) A number of large computer-based systems have been built which allow users to pose queries and retrieve data at remote terminals. Finally, recent work in computational linguistics has uncovered techniques which make it possible for the users of such systems to phrase their requests in a manner that is quite natural to them, and for the system to transform these requests into structures which can be efficiently matched against the contents of the information store.

---

<sup>†</sup> Hearings Before a Subcommittee of the Committee on Appropriations, House of Representative, Ninety-second Congress, First Session, Part 3, 1971, p. 1036.

Interview with Commissioner Robert Gottschalk, New York Times, January 28, 1972.

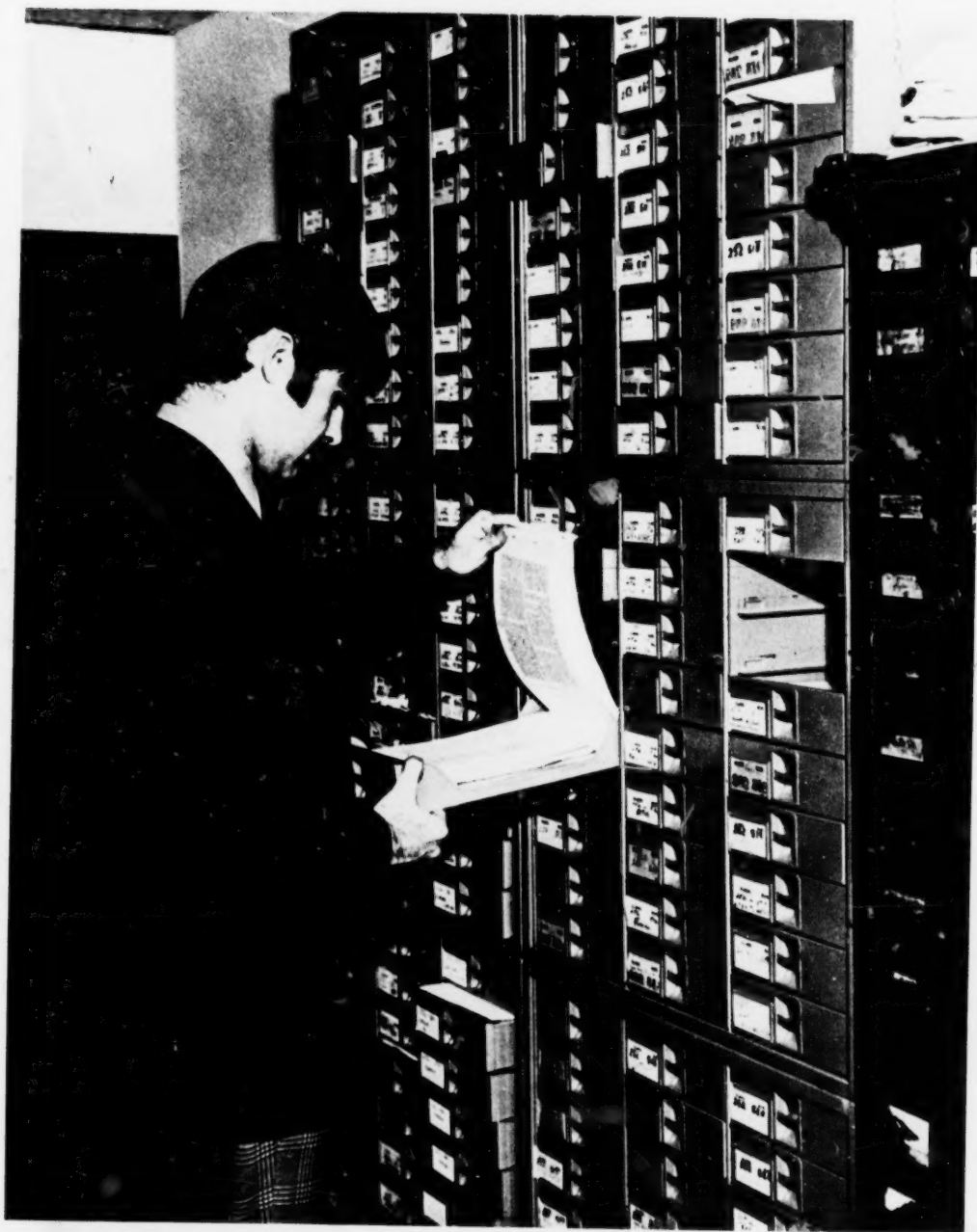


Figure 2. The manual system: searching through patents in the public search room (courtesy U.S. Patent Office)

## CHAPTER B

### THE WORK STATEMENT

#### REQUIREMENTS OF THE RFP

The Request for Proposal 1-35539 called for a design of an on-line query language for full-text patent search and for an operating interactive computer program which incorporates the most basic text-matching algorithms.

In particular, the query language was to be in quasi-English, generated by a grammar that facilitates parsing and syntax-error detection, and capable of specifying strings with at least as much power as that expressed by regular expressions with Kleene star events. Other capabilities included in the language were to be based on previous Patent Office studies of the characteristics of patent examiner search activity and on close consultation with Patent Office staff. While the design of the language was to be independent of any particular file organization and any particular programming language, the type of remote terminal was specified to be a display.

The Request for Proposal further stated that the end products of the design should be a report containing the formal description of the query language in terms of its syntax, constituents, rules of use, directed semantic action, and algorithms of translation. The body of this report constitutes the fulfillment of this requirement. Appendix II discusses the initial implementation of the design on a computer facility provided by the Patent Office.

## CHAPTER C

### OVERVIEW OF CAPABILITIES

The system which has been designed is oriented towards introducing to the patent examiner automated aids for searching patent text. In this context, the search process is viewed as an interactive one in which the examiner narrows the document collection to a manageable subset by (1) formulating a set of phrases (the search prescription) whose presence or absence in a document is purportedly indicative of its ability to "show" the invention, (2) reviewing the document texts so defined, (3) refining the search prescription to diminish, enlarge, or redefine the subsets, and so on. It is not the intention of the system design to guide the examiner to a perfect "102"; rather, the examiner should come away from the computer-assisted search with a list referencing a small subset of documents which are sufficiently promising to warrant more intense perusal at his desk.

As indicated above, there are two primary functions which support the subsetting process: searching and browsing. Searching takes the examiner's specification of a logical combination of patterns of words, word parts, and numerical values and finds those patents containing matching text. Any part of this search prescription can be restricted to finding a subpattern occurring entirely within one or more designated sections of a patent, or further, to within a paragraph or even a sentence. The search target is not confined to the original data base; it can also be the result of a previous search. Browsing takes the set of patents, or patent parts, and presents them on the display screen of the examiner's terminal in the order and in the manner the examiner directs. That is, the examiner may choose to look at some or all of matching texts or he may wish to view other text portions of those patents identified by the searching function (of course, he can always view the text of any patent in the data base, whether it was selected by a search function or not). Thus the user interacts with the system by performing searching and browsing in an order which is determined by his assessment of the success of his prior interactions.

This description of the examiner's convergence towards a relevant subset should not be construed as a uniform convergence. Intermediate search results may suggest a totally different approach than the one he had been following. The examiner can try a variety of paths to his goal, perhaps discarding an old way for a new one, perhaps making a brief excursion to test a hypothesis but returning to the original line of pursuit. The system design supports such digressions by retaining intermediate results.

In the course of convergence, the examiner has at all times access to whatever standard reference works have been made available to the system: class and subclass information, words-of-art with their meanings, lexicons, thesauri, administrative memoranda; plus tables generated by the system itself from patent data: type/token lists, match counts. The examiner himself can generate files and store them in the computer, analogous to his practice in the manual system: he can make annotations to patents, keep a notebook, establish his own lexicon and thesaurus; and he can limit access to these personally generated materials to himself alone, to the examining staff, or to the public-at-large as well. All of these auxiliary aids can be searched and browsed through just as if they were patents themselves.

The third primary function which supports the subsetting process is defining. The examiner can establish a set of synonyms for a word which are automatically searched for whenever that word appears in his search prescriptions. With this same capability, he can define, for any expression in the query language, an alternate which he feels is more mnemonic, easier to type, or just somehow more personally appealing. He can also define a single name for an entire statement in the query language, and even for a sequence of statements. The defining function is an extremely useful tool for handling the varieties of expressive mode found in natural language and for tailoring the language to individual users.

A pervasive influence throughout the design phase has been an awareness of the human element. The variety of user experience with the system is accommodated by providing plural modes of system-user interaction and plural dialects within the query language. The errant user is protected from inadvertently damaging the system or the data by extensive syntax checking and some semantic checking of his commands, and by password locking of system software. The user "in trouble" is given suggestive error comments and the capability of undoing history by restarting from a number of time points cited earlier in the console session. An emergency interrupt capability allows the correction of mistakes which so many times the user recognizes immediately afterwards.

One of the common complaints with interactive systems is the feeling of "distance" from the system. MITRE's design meets this objection by giving a response to every user action either confirming the execution of his request or calling his attention to a mistake and suggesting possible sources for the error. Furthermore, the status of the user vis-a-vis the system (waiting, executing, etc.) is constantly displayed on the screen, and the user has additional commands with which to query the status of the system.

Finally, for training or record-keeping purposes, a transaction history of the session is produced, at the option of the user.

## CHAPTER D

### PATENT STRUCTURE

Of course, a patent is not stored in the computer the same way as the printed copy. The most radical difference, as called for in the Patent Office specifications for the magnetic tape to be supplied to the system, is that all drawings and tables are culled out. Second, certain artifacts of the printing process, such as line numbers, column numbers, and page layout, although signaled in the computerized text by special codes, are ignored for purposes of search and display. However, meaningful elements such as bold face, superscripts, subscripts, and non-English alphabets are retained; and they are represented by special prefixes and suffixes attached to in-line, English analogues of such characters.

The patent is structured within the computer as a sequence of 32 data elements, or patent sections, such as date of application, attorney, abstract, claims, etc. Patent sections are further divided into paragraphs, within these, sentences, and within these, words. The data base supplied by the Patent Office is specially marked as to section and paragraph; sentence and word, however, are defined algorithmically. Individual numbered claims are considered to be paragraphs.

It is important for the examiner to be aware of this structure because he can browse through a patent by leaping from structure-type to structure-type and he can restrict the machine to search at specified structure levels. That is, the examiner can ask to see the next paragraph or he can request a search for a pattern which must occur entirely within a paragraph. However, he cannot ask to see the next column or restrict a search to a pattern occurring entirely within a column.

Each patent section has a name. The complete list can be found in an appendix to this report; it is sufficient now to list the primary data elements of interest to the examiner:

- title
- abstract
- brief summary
- drawing description
- detailed description
- claims

In our preliminary interviews with several patent examiners, we found no agreement as to terminological usage. Thus, the design of the

system allows several names to be synonymously associated with any data element, as long as no ambiguity is introduced. Some examiners, for example, will feel more comfortable with the label "object" instead of "brief summary ". Futhermore, any subset of data elements can be given a collective name; for example, the Patent Office has suggested the name "text" for the agglomeration of the six patent sections listed above. All sections of a patent are searchable and displayable.

## CHAPTER E

### QUERY LANGUAGE STRUCTURE AND CONVENTIONS

#### THE BASIC COMMANDS

The query language centers around three commands:

*find*

*show*

*let*

With a basic knowledge of these three commands (plus the simple procedures to log on and off the system), an examiner can immediately begin to do profitable work on the system. As he gains experience and becomes comfortable with the system, he can be introduced to powerful extensions of these commands and to other commands with which he can increase his skill and proficiency.

The *find* command performs the search function, that is, it specifies the complex pattern of words and stems to be searched for in the data base. The *show* command fulfills the browsing function; it displays the results of the search in a variety of ways and allows browsing through the original data base and through derivative texts formed by the search mechanism. The *let* command defines substitution classes (classification), forms routine command sequences, and marks the point of departure for excursions.

#### MODES OF INTERACTION

There are two modes of interaction between the user and the system: prompting and anticipatory. In the prompting mode the examiner directs system action by replying to questions displayed on the terminal. In the anticipatory mode the examiner directs the system action on his own initiative -- in effect, answering the sequence of questions in advance. There is no dichotomous choice between these two modes of interaction: if the examiner chooses not to anticipate the next question, or more likely, if he is distracted from the course of action, the system will prompt for the information it needs. These two modes will be discussed in more detail in a subsequent chapter.

## DIALECTS

There are also two dialects of the query language, one for the novice user and one for the more experienced user. The novice dialect is English-like in expression; the other dialect makes heavy use of symbolic abbreviations and "understood" meanings for missing constituents. The novice user can be taught a subset of the language and then operate by using only some of the system's full capability or by using default conditions for those capabilities of which he is not yet aware. There is no dichotomous choice between dialects: the examiner can employ expressions from either dialect as his own experience matures and he incrementally learns the system. For example, the following search request is rendered completely in the novice dialect:

*find all patents on "file with (pears or prunes in sentence in abstract) and (pitted in claims)*

The equivalent request, completely abbreviated, is:

*find (pears|prunes/s/ab) & (pitted/cl)*

A mixed dialect rendering is exemplified by:

*find (pears|prunes in s/abstract) and (pitted/claims)*

Thus the user can express himself in a fashion with which he feels most comfortable and at the same time bootstrap himself up to more abbreviated means of expression.

## COMMAND STRUCTURE

Commands consist of a command name followed by one or more arguments. If a sequence of commands are to be entered at one time, they must be separated by the command terminator, an exclamation point.

Arguments to commands may be separated by the argument terminator, a semi-colon. If the input message ends with the argument terminator, the system will prompt for missing arguments. What constitutes an argument to a command is a property peculiar to each command.

Some commands employ dedicated words or symbols, termed reserved words, to separate or to identify their arguments or

argument constituents. When these words or symbols identify the nature of the component to follow, they are called operators; when they merely serve to delineate, they are called noise words. Operators may have different meanings in different commands, if the interpretation is a natural one.

The examiner need not concern himself with spacing. If the specification of a command allows or requires a single space between certain constituents in its argument, then any number of spaces is permissible; e.g., *find dried plums* is equivalent to *find   dried plums*.

When the user wishes to use a word or symbol in an argument which happens to be spelled the same as a valid reserved word for that command but which is not intended to serve as a reserved word, he must make a literal out of that word or symbol. This is done by enclosing the element in single quotes. An objective of the query language design has been to choose operators and keywords carefully so that the need for literals is very small, although in some instances the need for literals is unavoidable; e.g., *find 'and' or 'or' gate*.

Certain system variables are referenceable by the user. These are denoted in the query language by a double quote prefix; e.g., *"search, "result, "state*.

Entities defined by a user (with the *let* command) and peculiar to him are denoted by a dollar sign prefix; e.g. *\$fruit, \$stdsearch*.

#### IMPLEMENTATION CONSIDERATIONS

At some point in the design of a query language, the implementation must be taken into consideration. Of all the external influences, the choice of remote terminal has had the most impact. The Patent Office selected a Computer Optics keyboard/display unit with, inter alia, built-in editing capabilities, large capacity screen (30 X 100), upper and lower case characters, and line addressability (so that the unit can act as a true two-dimensional device, as opposed to a one-dimensional teletypewriter or a display mimicking a teletypewriter). The terminal is tied to the CPU by high-speed communication lines. These features of the configuration influenced the language and the overall system design as detailed below:

##### Display Screen Division

The ability to address any line of the display for reading and writing allows us to dedicate fixed portions of the screen to particu-

lar uses, thereby assuring coherent presentations of tabular material and minimizing user eye movement. (In a one-dimensional display, (1) the point of attention continually shifts downward, jumping to the top line after the bottom line has been reached, and (2) lines of display are not erased but left on the screen until overwritten. It is especially annoying to maintain one's bearings, so to speak, in such a mode of display.) Our design divides the face of the display horizontally into four major areas: system status (2 lines), user input (2 lines), system response (2 lines), and data (remainder). System responses which are too long to fit the response block will be displayed in the data block.

#### Editing

The examiner will not infrequently wish to correct a miskeyed command. The user input will be retained until successful execution of the command so that misspellings, omitted terms, and the like can be corrected without extensive rekeying. Because the local editing capability of the terminal allows the examiner to modify what is on the display before transmitting it to the computer, all software editing functions can be eliminated from the query language.

#### Capitalization

Except for literals, case will be ignored. More particularly, commands may be input in any combination of cases; and although patent text will be displayed in both cases as in the printed document, searching will ignore case distinction unless specifically called for by a literal in the search prescription.

#### Messages

The large capacity screen and high speed communication lines allow the system to respond to the user with informative messages of some length, in contrast to the abbreviated messages found in systems with terminals operating at slow teletypewriter speeds. Meaningful error messages are especially important. The system design calls for a quadripartite format: message number / error condition / particular error detected / suggested or possible courses of action. The message number alone will be sufficient to remind the experienced user of the nature of his error; less experienced users will gain an understanding of their mistake by reading further along the error message.

#### Emergency Interrupt

It is absolutely necessary in an interactive system to provide a means for interrupting the system from a terminal. Usually the

terminal provides hardware which will generate an attention signal which automatically interrupts the computer. Because the selected terminal did not contain such a hardware feature, reserved word "@@" was set aside in the language for this purpose.†

---

†This is generally an unsatisfactory method for achieving interruption because it requires the supervisory program to periodically poll the terminal to see if there are any messages to be transmitted and then to detect the reserved word in the message. While all this is going on, the situation of concern may have deteriorated beyond repair. MITRE strongly recommends that the system be implemented with a terminal containing an attention key.

## CHAPTER F

### SEARCHING

#### SYNTAX OF THE COMMAND

The command to carry out the search function has two arguments: The source file to be searched and the prescription of the pattern of words and word parts to be searched for. A simplified version of the syntax is presented here. A complete description is given in the section on formal grammar.

*find SOURCE p*

The second argument, *p*, is the search prescription, to be defined later. The first argument, *SOURCE*, can be certain file names, such as:

*"file* -- the system name for the file of patents (data base) designated during log-on as the main file of interest for the session

*"result* -- the system name for the file (search product) referring to the patents identified in the most recent search

any file name coined by the examiner to denote a previously constructed data base or search product

If the examiner enters the command without the source argument, then *"file* is assumed; if he leaves out the prescription argument, a null search is made, yielding only the count of the number of patents in the source file; (if he leaves out both arguments, the system will prompt for them). To achieve an English-like flavor, the noise word *on* may be placed in front of the source file name, e.g.,

*find on "file specular mirror or reflector*

As just noted, the above command has these equivalent renderings:

*find "file specular mirror or reflector*

*find specular mirror or reflector*

The product of a search is a file which identifies those patents whose contents satisfy the search prescription. The examiner may find it profitable to visualize the search product as a newly created file

which contains only those patents.

Successive searches on search product files are not automatically kept. Hence a search on the search product file "*result*" will create a new search product file "*result*", the latter overwriting the first. The examiner may, however, retain intermediate search products by utilizing the definitional capabilities of the system (see Chapter G).

#### TABLE OF SEARCH CAPABILITIES

Listed below are phrases, in both dialects, exemplifying the full range of search capabilities provided in the search prescription, p, of the *find* command. Column one names the capability, column two provides one or more examples in the novice dialect, and column three shows the same capability expressed in abbreviated form.

| <u>CAPABILITY</u>                        | <u>NOVICE DIALECT</u>                       | <u>ABBREVIATED FORM</u>   |
|------------------------------------------|---------------------------------------------|---------------------------|
| CONJUNCTION                              | pears and plums                             | pears & plums             |
| DISJUNCTION                              | pears or plums                              | pears   plums             |
| CONCATENATION<br>with indefinite<br>span | fresh plums                                 | [no short form]           |
| PHRASING                                 | (dried plums) or<br>prunes                  | (dried plums)   prunes    |
| CONCATENATION<br>with exact span         | [no expression]                             | pears . . plums           |
| CONCATENATION<br>with bounded span       | pears 6 words plums                         | pears 6 plums             |
| OPTIONALITY                              | (dried plums) or<br>(dried purple plums)    | dried <purple>plums       |
| NEGATION                                 | plums and not dried                         | plums & ¬dried            |
|                                          | plums but not dried                         | plums & ¬dried            |
|                                          | plums but neither<br>dried nor frozen       | plums & ¬(dried   frozen) |
|                                          | plums but not dried<br>and not frozen       | plums & ¬dried & ¬frozen  |
| SEQUENCE                                 | frozen then plum then<br>pie within 6 words | frozen * plum * pie /6    |

| <u>CAPABILITY</u>               | <u>NOVICE DIALECT</u>                                             | <u>ABBREVIATED FORM</u>                                                                                        |
|---------------------------------|-------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| PERMUTATION                     | frozen with plum with<br>pie within 6 words                       | frozen , plum , pie /6                                                                                         |
| REPETITION                      | [no expression]                                                   | a yum % dessert                                                                                                |
| LOCAL SCOPE                     | pears and plums in<br>sentence<br>pears and plums in<br>paragraph | pears & plums /s<br>pears & plums /p                                                                           |
| GLOBAL SCOPE                    | pears and plums in<br>abstract                                    | pears & plums /ab                                                                                              |
| SCOPE ADJUNCTION                | pears and plums in<br>abstract plus title                         | pears & plums /ab + ti                                                                                         |
| INTERVAL                        | gear and from 32 to<br>46 teeth                                   | gear & 32 to 46 teeth                                                                                          |
| WORD INTERNAL                   | benzene or benzine<br>mcdonald or macdonald                       | (benz e i ne/w)<br>(m <a> cdonald/w)                                                                           |
| STEM<br>with indefinite<br>span | [no expression]                                                   | froz*                                                                                                          |
| STEM<br>with exact span         | [no expression]                                                   | theat..                                                                                                        |
| STEM<br>with bounded span       | [no expression]                                                   | freez3                                                                                                         |
| COMMENT                         | [no expression]                                                   | freez3 [will find <i>freeze</i><br><i>freezing</i> <i>freezer</i> <i>freezes</i><br>but not <i>freezable</i> ] |
| SYNONYM                         | [no expression]                                                   | \$fruit pie                                                                                                    |

#### EXAMPLES OF SEARCH PRESCRIPTIONS

While the syntax of the search prescription is quite complex because of its power and flexibility, it is actually quite easy to use. This is illustrated by the examples below, in which the meaning can usually be deduced from the form of the language and the table of capabilities previously presented.

For each example below, a rendering is given in the novice dialect where possible and also in the abbreviated form which a more experienced user might employ. Most of the examples should be self-explanatory. Note that relatively simple search prescriptions are expressed straightforwardly in the novice dialect. Complex search prescriptions are not so easy to render in that dialect; but on the other hand, the novice will not be composing complex patterns.

- (1) *find pulse or digital in claims*  
*find pulse | digital/cl*  
asks for either word to be present in the claims section
- (2) *find ultrasonic delay line and stagger-tuning within sentence in briefsummary*  
*find ultrasonic delay line & stagger-tuning /s/bs*  
asks for both phrases to occur in the same sentence of the brief summary
- (3) *find capacitive with read-only with memory within 8 words within paragraph in detaileddescription*  
*find capacitive, read-only, memory/8/p/ded*  
asks for the three words to occur within eight words of each other, in any order, provided that they also occur in the same paragraph of the detailed description
- (4) *find capacitive then read-only then memory within 8 words within paragraph in detaileddescription*  
*find capacitive \* read-only \* memory /8/p/ded*  
asks for a phrase, not more than 10 words long and all in the same paragraph of the detailed description, beginning *capacitive* and ending *memory* with *read-only* anywhere in between
- (5) *find (quartz in detaileddescription ) and (ultrasonic 2 words delay line in abstract plus briefsummary)*  
*or (ultrasonic delaying and means within paragraph in claims)*  
*find (quartz /ded) & (ultrasonic 2 delay line/ab+bs)| (ultrasonic delaying & means /p/cl)*  
no more than two words can appear between *ultrasonic* and *delay*
- (6) *find (implement or device or means or assembly) and (punch\* or pierc\* or perforat\* or punct\* with stripping within 4 words in abstract plus briefsummary plus detaileddescription*  
*find implement|device|means|assembly & (punch\* | pierc\* | perforat\* | punct\*, stripping/4) /ab+bs+ded*

- (7) *find pear and pomegranate and (plum or prune) and peach  
but neither orange nor lemon*  
*find pear & pomegranate & plum|prune & peach & ¬(orange|lemon)*
- [comments are placed between square brackets; nothing between such brackets has any effect on the search; an equivalent rendering of the above command is] find pear and pomegranate and (plum or prune) and peach and not orange and not lemon*
- [in the abbreviated dialect] find pear & pomegranate & plum|prune & peach & ¬orange & ¬lemon*
- (8) *find pentode or (  
'5' or five) element 2 words (electronic or therminonic)  
(tube or valve)*  
*find pentode | ('5' | five element 2 electronic|therminonic  
tube|valve)*  
*[or equivalently] find pentode | ('5' | five element 2 (electr|  
thermi onic/w) tube|valve)*
- (9) *find ( ((photosensitive substrate) or (photosensitive glass  
substrate)) with deposition within 10 words) and ( (evap-  
oration masks) then (printed circuits) within 2 words)*  
*find photosensitive <glass> substrate, deposition /10 \**  
*evaporation masks 2 printed circuits*  
*the phrase photosensitive ceramic substrate will not satisfy  
the first conjunct of this search prescription*
- (10) *find (voice 4 processing) & (frequency division . 2 terminals)  
& (solid state . pass filters)*  
*0 to 4 words must appear between the words of the first phrase,  
1 to 3 words between the words of the second phrase, and  
exactly 1 word between the words of the third phrase*
- (11) *find vacuum chamb.. & photosens\* & ferr3 oxide*
- (12) *find refractive index and from '.63' to '.94'*  
*find refractive index & '.63'to '.94'*

#### OPERATOR PRECEDENCE

How does one read

*pears and plums or prunes* ?

Is it

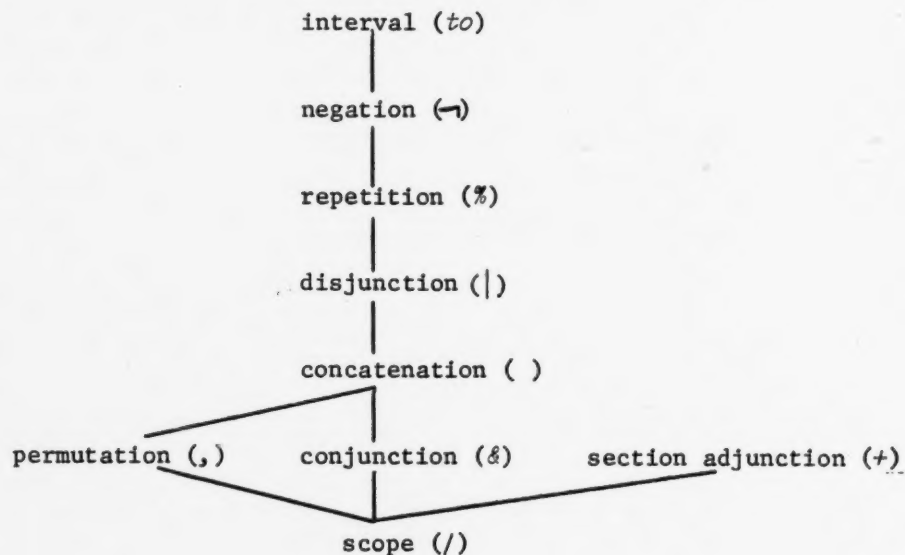
*(pears and plums) or prunes*

or

*pears and (plums or prunes)*

?

Most operators in the search prescription of the query language have a precedence which determines whether parentheses are needed for grouping. The precedence is shown below, where operators higher in the list bind before ones lower in the list:



The mirrored pair operators (parentheses, square brackets, and angle brackets) by their very nature do not participate in a precedence ordering.

Under this precedence among operators, the grouping *pears and (plums or prunes)* may be written

*pears & (plums | prunes)*

or

*pears & plums | prunes*

but the grouping *(pears and plums) or prunes* must be expressed with parentheses as

(pears & plums) | prunes

The following examples further illustrate the use of precedence:

| <u>search prescription</u>            | <u>interpretation</u>                                                                                               |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <i>find pulse digital/cl</i>          | <i>pulse or digital in the claims section</i>                                                                       |
| <i>find pulse (digital/cl)</i>        | <i>pulse anywhere in the patent or digital in the claims</i>                                                        |
| <i>find square wave pulse</i>         | <i>square wave or square pulse</i>                                                                                  |
| <i>find square (wave &amp; pulse)</i> | <i>both square wave and square pulse</i>                                                                            |
| <i>find square * wave pulse</i>       | <i>any string beginning with square and ending with either wave or pulse</i>                                        |
| <i>find ¬square pulse &amp; wave</i>  | <i>both wave and any two word phrase beginning with a word other than square and ending with pulse</i>              |
| <i>find ¬ '3' to '5' centimeters</i>  | <i>outside the range of 3 to 5 centimeters</i>                                                                      |
| <i>find degenerate ¬square% wave</i>  | <i>any string longer than two words, beginning with degenerate, ending with wave, and no word square in between</i> |

#### FORMAL GRAMMAR OF THE SEARCH PRESCRIPTION

The search prescription is the one component of the query language which is sufficiently complex in its syntax to require a context-free grammar and an associated parsing algorithm. (Decoding of the other commands can be done with a simple finite-state mechanism.) For several reasons, an operator precedence grammar† (OPG) was chosen to generate the search prescription language: (1) the Boolean nature of search requests lends itself to operator treatment, (2) the grammar is easily

---

† R. W. Floyd, "Syntactic Analysis and Operator Precedence", J. of the Association for Computing Machinery 10 (July 1963) 316-333.

modified as suggested by experience with an operating version of the system, (3) the most natural parser for an OPG is a bottom-up device, which provides superior error detection over top-down mechanisms in the sense that errors are detected at a point in the analysis at which meaningful error messages can be transmitted to the examiner, and (4) OPG parsers are typically table-driven, which means that syntax errors are immediately obvious to the language designer as gaps in the table, which can be filled by calls to error analysis and reporting routines.

Futhermore, a two-sided OPG was chosen to reduce possible examiner confusion between the allowable combinations of terms searched for in the text and allowable combinations of terms specified as the scope of the search.

What follows is the grammar as a formal set of production rules. The notation employed is lower case roman for non-terminal symbols, upper case roman for terminal symbol classes, lower case *italic* for the terminal symbols themselves,  $\Lambda$  for the null terminal symbol, and  $\vdash$  and  $\dashv$  for the meta-symbol string delimiters. The standard conventions of arrow for symbol rewrite and braces for rule disjunction are followed. The null symbol in rule disjunction with another terminal symbol denotes semantic equivalence, e.g.,

$$\text{SOURCE} \rightarrow \left\{ \begin{array}{c} \text{"file"} \\ \Lambda \end{array} \right\}$$

means that if no source is specified, then *"file"* is assumed by default. We distinguish terminal classes from the terminal symbols contained therein for reasons of efficiency in analysis of the search prescription. The assignment of symbol to class can be done by a simple lexical scanner before the complex mechanism of an OPG parser need be encountered.

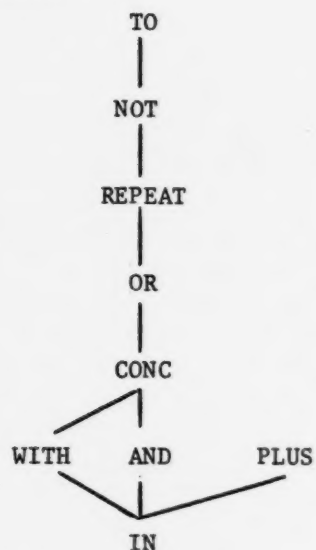
Command:

*find* SOURCE p

Non-terminal Rules:

1.  $p \rightarrow \left\{ \begin{array}{l} \vdash g \text{ IN } s \vdash \\ \vdash pd \vdash \end{array} \right\}$
2.  $pd \rightarrow \left\{ \begin{array}{l} pd \text{ AND } pd \\ pr \end{array} \right\}$
3.  $pr \rightarrow \left\{ \begin{array}{l} (pd) \\ pr \text{ OR } pr \\ pt \end{array} \right\}$
4.  $pt \rightarrow \left\{ \begin{array}{l} (pd) \\ NOT \text{ } pt \\ x \end{array} \right\}$
5.  $x \rightarrow \left\{ \begin{array}{l} (xd \text{ IN } s) \\ xd \end{array} \right\}$
6.  $xd \rightarrow \left\{ \begin{array}{l} xd \text{ AND } xd \\ xr \\ g \end{array} \right\}$
7.  $xr \rightarrow \left\{ \begin{array}{l} (xd) \\ xr \text{ OR } xr \end{array} \right\}$
8.  $g \rightarrow \left\{ \begin{array}{l} g \text{ WITH } g \text{ IN JOKER} \\ g \text{ WITH } g \\ gc \end{array} \right\}$
9.  $gc \rightarrow \left\{ \begin{array}{l} (g) \\ (g \text{ IN WORD}) \\ gc \text{ CONC } gc \\ gc \text{ CONC } m \text{ CONC } gc \\ gr \end{array} \right\}$

Precedence:



10.  $gr \rightarrow \left\{ \begin{array}{l} (g) \\ gr \text{ OR } gr \\ ge \end{array} \right\}$
11.  $ge \rightarrow \left\{ \begin{array}{l} (g) \\ (g) \text{ REPEAT} \\ gt \text{ REPEAT} \\ gt \end{array} \right\}$
12.  $gt \rightarrow \left\{ \begin{array}{l} (g) \\ \text{NOT } gt \\ \text{FROM PWORD TO PWORD} \\ \text{PWORD} \end{array} \right\}$
13.  $m \rightarrow \left\{ \begin{array}{l} m \text{ CONC } m \\ j \end{array} \right\}$
14.  $j \rightarrow \left\{ \begin{array}{l} (m) \\ (g) \\ \text{JOKER} \\ . \end{array} \right\}$
15.  $s \rightarrow \left\{ \begin{array}{l} \text{PARASEN} \\ \text{PARASEN IN } n \\ n \end{array} \right\}$
16.  $n \rightarrow \left\{ \begin{array}{l} (n) \\ n \text{ PLUS } n \\ \text{SECTIONNAME} \end{array} \right\}$

Terminal Classes [assigned by the lexical scanner]:

17. FROM  $\rightarrow \left\{ \begin{array}{c} from \\ \wedge \end{array} \right\}$
18. TO  $\rightarrow to$
19. NOT  $\rightarrow \left\{ \begin{array}{c} not \\ neither \\ \neg \end{array} \right\}$
20. REPEAT  $\rightarrow \%$
- 21.. OR  $\rightarrow \left\{ \begin{array}{c} or \\ | \end{array} \right\}$
22. AND  $\rightarrow \left\{ \begin{array}{c} and \\ but \\ \& \end{array} \right\}$
23. WITH  $\rightarrow \left\{ \begin{array}{c} with \\ , \end{array} \right\}$
24. IN  $\rightarrow \left\{ \begin{array}{c} in \\ within \\ / \end{array} \right\}$
25. PLUS  $\rightarrow \left\{ \begin{array}{c} plus \\ + \end{array} \right\}$
26. AND NOT  $\rightarrow \left\{ \begin{array}{c} nor \\ then \end{array} \right\}$
27. JOKER  $\rightarrow \left\{ \begin{array}{c} * \\ \text{INTEGER} \end{array} \right\} \left\{ \begin{array}{c} \text{WORD} \\ \wedge \end{array} \right\}$
28. INTEGER  $\rightarrow \left\{ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ \text{[and so on]} \end{array} \right\}$

29. PWORD→

[the definition of a "word" for patent search purposes will be supplied by the Patent Office; to this word may be attached a suffix (asterisk, digit or sequence of dots) which has a meaning at the character level analogous to the meaning of JOKER (rule 27) at the word level (cf. examples 10 and 11 in the earlier section, Examples of Search Prescriptions, the examples in the Table of Search Capabilities, and rules 13 and 18 in the next section Examples of Search Prescription, the examples in the next section, Intermediate Form of the Arguments); any sequence of characters not meeting this definition of a suffixed PWORD must be made a literal in order to be treated as a PWORD; a PWORD which is a literal may not take a suffix]

30. WORD→

$\left\{ \begin{array}{l} \text{words} \\ \text{word} \\ \text{w} \end{array} \right\}$

31. PARASEN→

$\left\{ \begin{array}{l} \text{paragraphs} \\ \text{paragraph} \\ \text{par} \\ \text{p} \\ \text{sentences} \\ \text{sentence} \\ \text{sen} \\ \text{s} \end{array} \right\}$

32. SECTIONNAME→

$\left\{ \begin{array}{l} \text{title} \\ \text{ti} \\ \text{abstract} \\ \text{ab} \\ \text{briefsummary} \\ \text{bs} \\ \text{drawingdescription} \\ \text{drd} \\ \text{detaileddescription} \\ \text{ded} \\ \text{claims} \\ \text{cl} \end{array} \right\}$   
 [see Appendix I for the complete list of lexical items in this class]

33. SOURCE  $\rightarrow$   $\left\{ \begin{array}{l} \text{on} \\ \wedge \end{array} \right\} \left\{ \begin{array}{l} \text{"file"} \\ \wedge \\ \text{"result"} \\ \text{[any user-coined filename]} \end{array} \right\}$

#### INTERMEDIATE FORM OF THE ARGUMENTS

The arguments to *find* are translated from their external form, suitable for human use, into an intermediate form, suitable for machine use.

The following pages contain the formal description of the search command at the intermediate level of the query language. For reasons of tradition, the syntax is expressed in Backus Normal Form, with  $\dagger$  denoting one or more occurrences of the component to which it is suffixed and with FULL CAPS denoting terminal symbols. The symbol ... denotes an open-ended, or at least a very large, set of disjuncts. Certain constituents are of a nature which requires consideration of implementation realities and hence are not defined.

High-Order BNF (above word level)

[meta-rule: constituents are distinguishable; e.g., in rule 16, <interval> is actually written as, say INTERVAL(305,460) or INTERVAL (305 460), not as INTERVAL (305460)]

1. <search request>::=<file name><sp1>
2. <sp1>::=<sp2>|<logical operator>(<sp1>↑)|SECTION(<section list>  
<sp2>)
3. <sp2>::=<sp3>|<logical operator>(<sp2>↑)|PARAGRAPH(<sp3>)
4. <sp3>::=<sp4>|<logical operator>(<sp3>↑)|SENTENCE(<sp4>)
5. <sp4>::=<string>|<logical operator>(<sp4>↑)
6. <section list>::=<section name>|OR(<section name>↑)
7. <section name>::=TITLE|AUTHOR|ABSTRACT|CLAIMS|...
8. <logical operator>::=AND|OR|NOT
9. <string>::=<word>|<interval>|<string operator>(<string>↑)|  
<string span operator>(<span><string><string>↑)|  
|CONCATENATION(<string><concarb>↑)
10. <concarb>::=<string>|OPTIONAL(<string>↑)
11. <string operator>::=OR|NOT|REPEAT
12. <string span operator>::=PERMUTATION|SEQUENCE
13. <span>::=<dot>|\*|<integer>|<dot><span>
14. <interval>::=INTERVAL(<integer><integer>)

### Low-Order BNF (word level and below)

[meta-rule: constituents abut; e.g., <digit>↑ would yield 5649]

- +15. <file name> ::=

---

<sup>†</sup> implementation dependent

16.  $\langle \text{word} \rangle ::= \langle \text{stem} \rangle | \langle \text{stem} \rangle \langle \text{suffix} \rangle | \langle \text{literal} \rangle$
- +17.  $\langle \text{stem} \rangle ::=$
18.  $\langle \text{suffix} \rangle ::= \langle \text{dot} \rangle^+ | * | \langle \text{digit} \rangle$
19.  $\langle \text{dot} \rangle ::= .$
20.  $\langle \text{literal} \rangle ::= ' \langle \text{character} \rangle ^+$
21.  $\langle \text{integer} \rangle ::= \langle \text{digit} \rangle ^+$
22.  $\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$
- +23.  $\langle \text{character} \rangle ::=$

#### Predicate Semantics

AND is True if every one of its arguments is True

OR is True if any one of its arguments is True

NOT is True if none of its arguments is True; is False if the empty string is present

OPTIONAL is True if any one of its arguments is True or if the empty string is present

CONCATENATION is True if its arguments occur within the same paragraph and they are adjacent to each other in the same order in which they are specified as arguments to the predicate

PERMUTATION is True if its arguments occur within the same paragraph and if every one of its arguments is within  $\langle \text{span} \rangle$  words of each other.  $\langle \text{span} \rangle$  is assigned the value \* if no JOKER restriction is specified in the external form of the query language.++

---

+ implementation dependent

++ This predicate is a special case of AND restricted to the domain of Paragraph. It was not given the same external form (*with* vs. *and*) to capture the user-felt distinction between unordered terms within a phrase and joint presence of keywords in an entire document. In addition, the coalescence of these predicates would have made the grammar ambiguous.

SEQUENCE is True if its arguments occur within the same paragraph, if every one of its arguments occurs in the order specified, and if the first and last arguments occur within <span> words of each other.

REPEAT is True if its argument occurs repetitively and adjacently one or more times.

INTERVAL is True if the text word under examination falls within the closed interval defined by the first argument (low) and the second argument (high).

#### Semantics of Terminal Symbols

- is satisfied by any single character or any single word. The sole ambiguity generated by this definition is the case of dot surrounded by spaces. We will interpret that as specifying any single word rather than any single character.
  - \* is satisfied by an unlimited (from zero up) number of characters or words, except of course, as restricted by scope imposed at a higher level.
- <integer> or <digit> is satisfied by any number of words or characters, up to the limit specified by the numerical value.

#### CONVENTIONS AND DEFAULTS

The symbolic operators & | ( ) [ ] / ~ , % < > + need not have spaces surrounding them; e.g., *find pears | plums* is equivalent to *find pears|plums*.

Repeating ourselves somewhat from an earlier section, literals are denoted by single quotes; e.g., *find '\$pattern'* initiates a search for the word *\$pattern*; *find 'and' or 'or' gate* initiates a search for the phrase *and gate* or *or gate*. The standard convention of doubling applies to making a literal of ' itself; e.g., *D'Arsonval*.

To search for words graphemically identical to reserved words in the *find* command or to search for character sequences which are not members of the PWORD terminal class, they must be made literals. (A more sophisticated parser could relieve, in part, this burden; but the speed of an unencumbered parsing algorithm and the uniformity of external representation to the user overpower elegance in this case.) An example of the former case has already been illustrated

with *find 'and' or 'or' gate*. Another example of this same case is *find '2'|(a pair) 2 cutting heads*, where the first '2' stands for the numeral and the second '2' stands for up to two joker words (i.e., no words, any word, any two-word phrase). An example of the latter situation is *find 'cutting head,'* which asks for the two words *cutting* and *head*, separated by exactly two spaces, and with a comma after *head* (assuming the punctuation marks space and comma are two of the ways of delineating a PWORD).

The same convention also holds for words containing certain other special symbols, viz.:†

askuser

@@

(

)

[

]

\$

"

;

!

,

For example, *find '5!' or '\$6.32'*.

One need not literalize words graphemically identical to command names (except @@) or operators to other commands (which aren't also on the above list). That is, *find find* and *find be* are acceptable; one need not write *find 'find'* or *find 'be'*.

---

† Some of these special symbols have not yet been introduced; their use will be explained in subsequent chapters.

Capitalization is ignored unless it occurs as a literal; e.g., *find moly\* bolt* is equivalent to *find MOLY\* BOLT*. The search initiated will attempt to retrieve the pattern no matter what case appears in the data base. *Find 'Moly'\* bolt* will restrict hits to initial cap *Moly\**.

Certain values, if missing, will be supplied by default by the system. It can be decided in implementation whether these values should be inserted by the query language translator or whether they should be designed into the search algorithm itself. For example, the command *find prunes|plums* is really searched as if it were expressed *find prunes|plums|patent*. In this particular case we chose the latter procedure in our description of the intermediate form of the language (cf. rule 2).

Section names joined by the PLUS operator at the external level (translated to a section list at the intermediate level) act as an indivisible unit during the search. For example, *find prunes & plums/ab+ti* is not equivalent to *find (prunes & plums/ab)|(prunes & plums/ti)*; i.e., only the former will be satisfied with a patent containing just *prunes* in the abstract and just *plums* in the title, or vice versa.

#### SUBTLETIES TO BE AWARE OF

The definition of a PWORD to be supplied by the Patent Office will have to be such that ordinary punctuation in the data base is non-interfering. That is, a request to find the word *gear* should match any of these character sequences in the patent: *gear*, *gear.*, *gear?*, *gear-*, *-gear*, *gear,*, *gear's*, *"gear"*.

Optional elements can only occur embedded within a concatenation: *<dried> plums* is an ungrammatical search prescription; *purple <dried> plums* is acceptable. N.B.: *find purple <dried> plums* will find *purple plums* or *purple dried plums* but not *purple frozen plums*.

The predicate NOT can be tricky to use correctly, since the common interpretation of the English word "not" is an amalgam of "nowhere", "not precisely here", and "anything else but". We have adopted a definition which has the following consequences: (1) *find ¬ plum* will find all patents which do not contain *plum* anywhere in them; (2) *find purple ¬ dried plum* will find all patents which contain a three-word phrase beginning *purple*, ending *plum*, and whose middle word is not *dried*; (3) *find purple <¬dried> plum* will find all patents which contain the phrase *purple plum* or the three-word phrases just described; (4) *find ¬ (pear & plum)/par* will find all patents which contain a paragraph without both *pear* and *plum* in it; (5) *find ¬(pear & plum/par)* will find all patents which contain no paragraphs with *pear* and *plum* in them; and (6) *find ¬(¬pear & plum)/par* will find all patents in which every paragraph contains both *pear* and *plum*.

The "optional" mechanism is similarly used with the REPEAT predicate to capture the "zero" case; e.g., *find purple dried % plum* will find all patents containing *purple dried plum*, *purple dried dried plum*, etc.; *find purple <dried %> plum* will find all patents which contain the phrase *purple plum* or the family of phrases just illustrated. Thus, an optional REPEAT corresponds to the classic Kleene star operator.

The examples presented in the earlier section, EXAMPLES OF SEARCH PRESCRIPTIONS, were, for pedagogical reasons, more strictly constructed than necessary. Extra parentheses were used, a distinction between *in* and *within* could be inferred (the grammar really allows these reserved words to be used interchangeably), and needless restrictions in scope were specified (examples 3 and 4 do not need the /p).

#### TRANSLATION PROCEDURE

The translation of the search prescription from the external form used by the examiner to the intermediate form used by the search program is a five-stage procedure:

- (1) dollar expansion -- replaces user-coined names with their string definitions†

---

†Described in Chapter G.

- (2) segmentation -- delimits PWORDS, reserved words, and comments
- (3) category assignment -- discards noise words, assigns syntactic categories, and inserts explicit concatenation operators for the implicit form employed in the query language
- (4) parse -- builds labelled tree to represent syntactic structure of the search prescription
- (5) prune -- flattens and optimizes the resultant tree and reconfigures it somewhat to accommodate the search program (if necessary)

All except step (5) can be carried out by finite state mechanisms and need no further comment. The nature of the grammar determines the degree of complexity required of the parser. Our grammar was chosen to be an LR(1) and OP context-free grammar<sup>†</sup> to enable us to take advantage of published proven techniques for parsing and error detection.<sup>††</sup> The grammar is further restricted to contain no rules with identical right parts, which makes it amenable to further techniques, such as the so-called augmented operator grammar of Gries. Almost all of the techniques for parsing statements generated by OP grammars are driven by a two-(or three-) dimensional table which exhibits the precedence relations between operators. The table can be produced automatically from the formal rules as we have given them.<sup>†††</sup>

---

<sup>†</sup> D. E. Knuth "On the Translation of Languages from Left to Right" Information and Control 8(Oct 1965) 607-639.

R. W. Floyd op. cit.

<sup>††</sup> J. M. Foster Automatic Syntactic Analysis American Elsevier, 1970, pp. 31-43.

D. Gries Compiler Construction for Digital Computers Wiley, 1971, pp. 121-153.

<sup>†††</sup> A computer program to accomplish this has been written, for another project at MITRE, by W. Gulden. It is more than likely that such a program has been written elsewhere.

Typically, the table is quite sparse; but since the empty cells represent "impossible", or ungrammatical, symbol groupings, we can make use of them as error detectors. By themselves, these empty cells, together with their coordinate table entries, can be used to automatically generate error messages. However, many of the individual error conditions can be grouped (by hand) into larger families, thereby reducing the apparent (to the user) number of error situations to a manageable subset. Each of these families can have a suggested correction associated with it and therefore can be of more value to the user. (This technique was utilized in the initial implementation of the system, as described in Appendix II.)

#### NORMAL AND ERROR RESPONSES

The normal response to a search request is a confirmatory message in the response area of the display stating that searching has begun. The data area will contain a recapitulation of the search request, retaining the upper/lower case distinctions made by the examiner and expanding all substitution labels (see Chapter G on "synonym" definitions), followed by a table of frequencies of occurrence for each PWORD in the search prescription. The entries in the table shall follow the same order as in the search prescription, even to the point of duplication of entries if words are used repeatedly in the search prescription. For each entry the table will list the number of documents in the file containing that word and the number of times that word occurred in the entire file of documents (for stems, only a lower bound may be available; and that is satisfactory if so marked). At the completion of a search, the response area is rewritten with an appropriate message, the number of documents found is appended to the other information present in the data block, and the input block is erased.

The response to a grammatically incorrect search request is a message in the response area calling attention to the fact that a syntax error has occurred and requesting the examiner to correct his command and try again. The data area will contain an error message formatted as follows: error number / type of error / a short statement explaining the perceived error and suggesting a correction or alternative corrections / a recapitulation of the expanded search request marked to identify the point at which the error begins. (This information can be generated directly from the augmented precedence table and other structures of the parsing mechanism.) Note that the error message becomes more detailed as the user reads further. In this manner, the errant user can stop reading as soon as he identifies the problem which, of course, varies with the experience of the user and the nature of the syntax error. We refrain from spelling out the myriad of error messages here; the reader can appreciate the intent of our design from the three examples below:

INPUT: find plums and /ab

RESPONSE: Sorry, cannot understand your search request;  
please correct and try again

DATA: F18 Syntax Error  
An IN cannot directly follow an AND -- did  
you forget an argument?

plums and <==Error?==> /ab

INPUT: find plums & prunes|par

RESPONSE: Sorry, cannot understand your search request;  
please correct and try again

DATA: F11 Syntax Error  
'PAR' is a scope name, and as such it cannot  
be part of the pattern searched for (without  
enclosing it in single quotes) -- did you  
mean to use an IN?

plums & prunes | <==Error?==> par

INPUT: find dried \$pfruit

RESPONSE: Sorry, cannot understand your search request;  
please correct and try again

DATA: F18 Syntax Error  
A right parenthesis cannot directly  
follow an AND -- did you forget an argument?

dried [\$pfruit] (plums and prunes and  
pomegranates and <==Error?==> )

Since the parser analyzes left-to-right, multiple ungrammaticalities will be detected and reported one at a time. This procedure insures the user that everything to the left of the marked error is "clean".

If a search request is grammatically correct, but one or more of the PWORDS have a zero frequency of occurrence, then the response shall be a message suggesting the presence of spelling errors and requesting the examiner to reword his command and try again. (Of course, a zero frequency of occurrence could just as well mean that the sought-after word really does not appear in the file; but a spelling error, which the examiner had never noticed under the manual system because of the predictive nature of reading and understanding natural language, is a severe impediment to computer-based systems and must be specially called attention to.) The data area will contain the expanded search request with zero-occurrence words marked and the table of frequencies.

A search request may be unprocessable because a referenced user-coined name has never been defined. The error response in this case is an appropriate message in the response area listing the undefined names and in the data area a recapitulation of the search request expanded to the extent possible.

In all cases, the input area is not erased until a search goes to full completion. This allows ready correction of an erroneous search request with the hardware editing features of the selected display terminal. The response and data areas are erased as soon as the monitor detects the input of a new command. This procedure leaves the information generated by the previous search request on the screen as a guide to formulating a new command to search or to browse, and at the same time the clearing of these areas signals the user that his new command has been "taken" by the system. Erasing the input area prematurely burdens the examiner with retyping the entire command, only part of which was erroneous; not erasing the response and data areas soon enough may confuse the examiner by the simultaneous presence of a new command and a response to an old command. Finally, in every case, the cursor will be reset to the input area.

#### NATURE OF THE SEARCH PRODUCT

The result of a search is a subset of the source file, termed "*result*", containing the entirety of all patents satisfying the search prescription. Other commands, to be discussed in subsequent chapters, will allow him to view and print the patent numbers of the patents just found, to locate exactly those paragraphs containing specified words, to put the members of "*result*" in chronological order and what have you.

The frequency table and other items in the data area are stored under the system name "*freq*" to allow subsequent recall and display if needed.

As mentioned in the beginning of this chapter, "*result*" and "*freq*" are transient files; they pertain only to the immediately prior search.

## CHAPTER G

### DEFINING

#### SYNTAX OF THE COMMAND

The command to carry out the define function has two arguments:

*let NAME be DEF*

where:

NAME is the user-chosen name for DEF

DEF is a string of characters which is substituted for NAME when NAME is prefixed by a dollar sign

#### EXAMPLES OF USE

As can be seen by the syntactic description above, the *let* command is simply a substitution mechanism. However, the unrestricted nature of the definiens makes the command a powerful one. Further, nested definitions can be set up by including dollar-prefixed names in the definiens, which ability lends itself to quite sophisticated use.

The examiner can use *let* to define concepts by agglomerating expressions of patent dialect or technological terminology which are synonymous for search purposes, e.g.,

*let hole be perforat\*|aperture|hole|puncture*

*let bistable be bistab\*|(two condition)|flipflop|flip|  
flop|relaxation|trigger*

The string expressions don't have to entertain pretensions of synonymy; they can be partial search prescriptions such as

*let a be cutting|slicing machine|device  
let a2 be cutting|slicing 2 machine|device  
let asen be cutting|slicing \* machine|device  
let ch be chopping|chewing*

which the examiner can assemble in various ways to make a completed search prescription:

```

find $a , heat treated , continuous feed|fed/10
find $a2 , heat treated , continuous feed|fed/10
find $ch|$asen 2 continuous feed /par
find not $ch and $asen in abstract

```

A potential problem with completely unrestricted substitution within a search prescription is unforeseen interrelation between the visible operators entered by the examiner and "hidden" operators present in definitions. Our design conservatively calls for the definiens to be parenthesized when the substitution is executed during the processing of a search request, thereby obviating the problem. An example of this situation is

```

let plums be prunes and plums
let grapes be raisins and grapes
find $plums or $grapes

```

The expanded search request has the interpretation the examiner probably intended, namely;

```

find (prunes and plums) or (raisins and grapes)

```

rather than the other grouping,

```

prunes and (plums or raisins) and grapes,

```

implied by an unparenthesized expansion.

To save each search product for subsequent comparison (the system automatically saves a search product (under the name "result") only until the next completed search), the examiner can assign names to these search products; e.g., †

```

find $a , heat-treated , continuous feed|fed/10! let answer1
be "result
find $a2 , heat-treated , continuous feed|fed/10! let answer2
be "result
find $ch|$asen 2 continuous feed/p! let answer3 be "result
find $ch & $asen /ab! let answer4 be "result

```

followed at a later time with, say,

```

find on $answer1 rotary

```

---

†The exclamation point separates commands. It will be explained formally in a subsequent chapter.

A fourth use of *let* is to set up a classification, or thesaurus; e.g.,

```
let tree be $deciduous|$evergreen
let evergreen be conifer*|$softwood
let deciduous be broadleaf*|$hardwood
let softwood be (white pine)| (red spruce)|hemlock
let hardwood be (white oak)| (hard maple)|(black walnut)|
               (white ash)
```

To simplify the input of command arguments at the terminal keyboard, the examiner can coin abbreviations for long words or frequent standard phrasings, e.g.,

```
let scm be silicon controlled rectifier
```

A number of operations can be chained together so that the sequence -- perhaps a routine one -- is performed by specifying only its name; e.g.,

```
let stdsch be find on "newfile $pollution! show titles pn!
print
```

Finally, the examiner can save the current state of the system under a name so that he can digress and then return to his point of departure; e.g.,

```
let point3 be "state
```

As noted several times earlier in this report, but repeated here for completeness, reference to these names in other contexts, i.e., not as names but for their meanings, or substitution values, is by employment of the dollar-sign prefix; e.g.,

```
find $scr
$stdsch
return to $point3
```

When the substitution is made, however, bracketed comments associated with definitions are not brought forward too. Thus, a lexicon with extensive scope notes can be referenced without cluttering up the display face with annotations (recall that in the chapter on searching, we pointed out that one of the first steps in the processing of a *find* command is the redisplay of the search prescription expanded (all \$ substitutions made) so that the examiner can view the actual pattern about to be searched for; and if he is dissatisfied, interrupt processing).

All definitions during a session are effective from time of coining until log-off.

#### OTHER COMMANDS ASSOCIATED WITH DEFINING

Definitions, particularly those intended for inclusion within search prescriptions, should be checked for circularity before use. The command

*test NAME*

inspects the definition of NAME for circularity and for unresolved reference. If NAME is not specified, the entire set of definitions is tested. The error check procedures incorporated in the search function will also detect these two problems, but at an inconvenient time.

The command

*delete NAME NAME NAME ...*

deletes the definitions in its argument. Definitions referenced by name in the definitions of deleted definitions are not deleted.

Occasionally the examiner may wish to assign a new name to an existing definition. This can be done with the command

*rename NAME to be NAME*

which corrects the lexicon and releases the old name for future use.

Occasionally the examiner may wish to assign a new definition to an existing name. The command sequence

*delete NAME! let NAME be DEF*

will accomplish this function.

After a sizeable number of names have been defined, the examiner will need recourse to listings of names, definitions, and of particular importance, an inverted list containing those definitions which contain a specific name in their definitions. The inverted list is crucial to the successful modification or deletion of existing definitions, since another definition of which the examiner is unaware may refer to the definition being altered. The commands to accomplish these functions are discussed in Chapter H.

## NORMAL AND ERROR RESPONSES

The normal response to the *let*, *delete*, and *rename* commands is a confirmatory message in the response area of the display repeating or restating the action requested. The normal response to *test* is a simple "O.K. -- no formal improprieties found".

Since *let* and *rename* both require two arguments, an error comment is made if only one argument is supplied.

The rules for name formation will be decided by the implementation. *let*, *delete*, and *rename* will report ill-formed names.

If the examiner attempts to assign a definition to a name which already has been defined, an error comment pointing out the pre-existing definition condition will be displayed in the response area; and the old definition will be presented in the output area.

If the examiner attempts to delete a definition, via *delete* or *rename*, which doesn't exist, an appropriate comment to that effect is displayed. Note that *delete*, being multi-argumented, can have both correct and error responses at the same time.

There are two error responses to *test*: one denoting circularity and giving the circular path, and the second denoting unresolved references and displaying the list of undefined names. The latter condition is not a serious error and can occur simultaneously with the normal response if all else is well.

## NATURE OF THE DEFINITION PRODUCT

The file of all definitions coined by the examiner, including those retained from prior sessions, has the system name "*def*". This file is structured, so that the examiner can ask to see just all the names on the file, or all the definitions, or the names of the substitution classes referenced within a particular definition, or the names of the substitution classes referenced within the definitions of those names, or the names of all classes defined in terms of a particular word, or the names of all classes defined in terms of those names, etc.

## CHAPTER H

### BROWSING

#### GENERAL CAPABILITIES

The examiner has numerous choices in how files can be displayed to him. He can specify:

1. the level of the file which is to be directly addressable; that is, is one stepping through file patent by patent, or paragraph by paragraph
2. the number of entities at that level to be shown; that is, all entities in the file, or just the first five of them
3. the number of logical levels, below the selected one, to be displayed (this has no meaning for flat files, like "*freq*")
4. a list of words to be marked in the text when they are displayed<sup>†</sup>
5. an order in which the selected entities should appear; that is, chronological, reverse chronological, or alphanumeric (the level used to order need not be the one which has been selected for display)
6. the direction (forward, backward) and relative position in the file of the next entity to be displayed
7. the alignment of an entity on the screen (for viewing and printing purposes)
8. the area on the screen in which the entity will be displayed

This chapter will discuss displays directed toward the data area of the screen, by the command *show*; displays directed elsewhere are discussed in the last chapter of this report.

---

<sup>†</sup>This feature is intended to aid the examiner in selecting synonymous expressions for reformulating his search request, on the assumption that such expressions are in the proximity of the marked words, and also to reassure a doubtful user that *find* worked correctly. It is likely that the simple search capability of *show* will be implemented by a linear scan of the text, which would be ponderously slow for the full data base but is satisfactory for the small amount of text displayed by *show*.

## SYNTAX OF THE COMMAND

The command to display a file in the data area has six arguments:

$$\text{show SOURCE} \left\{ \begin{array}{c} \text{INTEGER} \\ \text{ALL} \\ \wedge \\ \text{pw} \end{array} \right\} \text{LEVEL DEPTH} \left\{ \begin{array}{c} \text{ORDER} \\ \wedge \end{array} \right\} \text{MARK}$$

The first argument has the definition:

$$\text{SOURCE} \rightarrow \left\{ \begin{array}{c} \text{on} \\ \wedge \end{array} \right\} \left\{ \begin{array}{c} \text{"file"} \\ \text{"result"} \\ \wedge \\ \text{"def"} \\ \text{"freq"} \\ \text{[any user-coined or system} \\ \text{filename]} \end{array} \right\}$$

If the examiner enters the command without the source argument, this formulation takes on the default value of "result". We will adhere to the the same convention used in the *find* command: if only the source argument is specified, then a count of the number of highest-level entities (e.g., patents or definitions) is displayed; if no arguments are specified, the system will prompt for one of them. The remaining arguments are defined as follows:

$$\begin{aligned} \text{ALL} &\rightarrow \left\{ \begin{array}{c} \text{all} \\ * \end{array} \right\} \\ \text{pw} &\rightarrow \left\{ \begin{array}{c} \text{pw CONC pw} \\ \text{PWORD} \end{array} \right\} \\ \text{LEVEL} &\rightarrow \left\{ \begin{array}{c} \text{PATENT} \\ \text{n} \\ \text{DEFINITION} \\ \text{NAME} \\ \text{WORD} \end{array} \right\} \\ \text{ORDER} &\rightarrow \left\{ \begin{array}{c} \text{in} \\ \wedge \end{array} \right\} \left\{ \begin{array}{c} \text{reverse} \\ \text{foward} \\ \wedge \end{array} \right\} \left\{ \begin{array}{c} \text{chronological} \\ \text{chrono} \\ \text{alphanumeric} \\ \text{alpha} \end{array} \right\} \text{order by SECTIONNAME} \end{aligned}$$

$$\begin{array}{l} \text{DEPTH} \rightarrow \left\{ \begin{array}{c} \text{down} \\ \wedge \\ \text{up} \end{array} \right\} \quad \left\{ \begin{array}{c} |0| \\ \wedge \\ \text{INTEGER} \\ \text{ALL} \end{array} \right\} \quad \left\{ \begin{array}{c} \text{level} \\ \text{levels} \\ \wedge \end{array} \right\} \quad \text{DEFINITION} \rightarrow \left\{ \begin{array}{c} \text{defs} \\ \text{def} \end{array} \right\} \\ \\ \text{MARK} \rightarrow \text{marked pw} \quad \text{NAME} \rightarrow \left\{ \begin{array}{c} \text{names} \\ \text{name} \end{array} \right\} \end{array}$$

where PWORD, n, WORD, SECTIONNAME, and INTEGER are defined as in rules 29, 16, 30, 32, and 28, respectively, of the formal grammar of the search prescription. PATENT is defined in the obvious way later in this chapter. Of course, not all the values of an argument have meaning in any source file; the mechanism for detecting and reporting inappropriate values is described in a subsequent section on error handling.

Files other than the data base, search results, or user-coined definitions can be displayed, browsed through, or even searched. For brevity, we will limit our discussion to the latter two file types.

#### EXAMPLES OF USE -- SEARCH PRODUCT

The following examples illustrate the use of the *show* command to call up the "result" file for viewing. This command only brings up the first "frame", or scopeful, on the screen of the terminal; subsequent sections will treat the manner of paging through the remainder of the file. Where the examples contain pairs, the commands are equivalent.

- (1) *show on "result all patents*  
*show patents*  
asks for the entire search product file to be made available for display
- (2) *show on "result 10 titles*  
asks for the titles of the first ten patents in the file
- (3) *show on "result all titles plus abstract*  
*show ti+ab*  
asks for just the title and abstract of each patent
- (4) *show on "result all patents in reverse chronological order*  
*by patentdate*  
*show pat reverse chrono order by pd*

- (5) *show on "result all ti+ab+bs+drd+ded+cl marked prune plum  
pear  
show text marked prune plum pear  
when the patent text is displayed, the words prune, plum,  
and pear will be underlined*
- (6) *show patents marked \$wordlist  
the definitional capability has domain in all commands,  
including show*

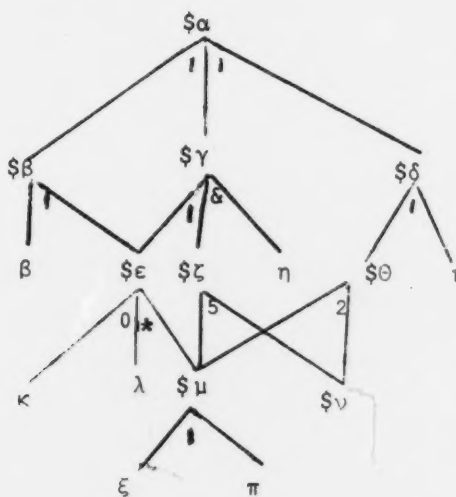
To facilitate browsing, accompanying all displays of patents or extracts of patents will be relevant data such as patent number, sectionname, and paragraph number.

#### EXAMPLES OF USE -- DEFINITION PRODUCT

Assume the following commands have been executed, where the Greek letters stand for any PWORD:

let  $\alpha$  be  $\$ \beta$  or  $\$ \gamma$  or  $\$ \delta$   
 let  $\beta$  be  $\beta$  or  $\$ \epsilon$   
 let  $\gamma$  be  $\$ \epsilon$  or  $\$ \zeta$  and  $\eta$   
 let  $\delta$  be  $\$ \theta$  or  $\iota$   
 let  $\epsilon$  be  $\kappa$   $\lambda$  \*  $\$ \mu$   
 let  $\zeta$  be  $\$ \mu$  5  $\$ \nu$   
 let  $\theta$  be  $\$ \mu$  2  $\$ \nu$   
 let  $\mu$  be  $\xi$  or  $\pi$

Then "def contains a representation of the structure:



The implementation would involve pointers to all the nodes in such a way that the structure could be readily traversed in any direction.

The following examples illustrate the use of *show* to call up the first "frame" of the selected portion of definition product files. (The characteristics of the display terminal will be a deciding factor as to whether such structured files are displayed by "indentation" or by directed graphs.) Where the examples contain pairs, the commands are equivalent.

- (1) *show on "def all names down 0 levels*  
*show "def names*  
 asks for all names defined by the user; i.e.,  
 the list  $\alpha \beta \gamma \delta \epsilon \zeta \theta \mu$  appears in the data area

- (2) *show on "def all names down 1 level*  
*show "def names 1*  
 asks for all names, and associated with them, the names in their definitions; thus an array matching the following tree representation will be displayed:

|                               |                  |                   |
|-------------------------------|------------------|-------------------|
| $\alpha(\beta \gamma \delta)$ | $\delta(\theta)$ | $\theta(\mu \nu)$ |
| $\beta(\epsilon)$             | $\epsilon(\mu)$  | $\mu$             |
| $\gamma(\epsilon \zeta)$      | $\zeta(\mu \nu)$ |                   |

The failure of  $\nu$  to have a definition is not discovered at this level.

- (3) *show on "def all names down 2 levels*  
*show "def names 2*  
 displays the trees:

|                                                                 |                   |                   |
|-----------------------------------------------------------------|-------------------|-------------------|
| $\alpha(\beta(\epsilon) \gamma(\epsilon \zeta) \delta(\theta))$ | $\epsilon(\mu)$   | $\underline{\nu}$ |
| $\beta(\epsilon(\mu))$                                          | $\zeta(\mu \nu)$  |                   |
| $\gamma(\epsilon(\mu) \zeta(\mu \nu))$                          | $\theta(\mu \nu)$ |                   |
| $\delta(\theta(\mu \nu))$                                       | $\mu$             |                   |

At this stage, the lack of definition for  $\nu$  is discovered, and  $\nu$  is marked

- (4) *show on "def all defs down 0 levels*  
*show "def defs*  
 asks for the list of definitions, essentially the eight  
*let* commands postulated earlier

- (5) *show on "def '\$\beta\$' def down 0 levels*  
*show "def '\$\beta\$' def*  
 asks for the definition of  $\beta$
- (6) *show on "def '\$\beta\$' names down 1 level*  
*show "def '\$\beta\$' names 1*  
 asks for the names in the definition of  $\beta$ ; namely,  $\epsilon$
- (7) *show on "def '\$\alpha\$' defs down all levels*  
*show "def '\$\alpha\$' def \**  
 asks for the definition of  $\alpha$ , the definitions of the names in that definition, the definitions of the names in those definitions, and so on; i.e., the structure for  $\$ \alpha$  presented at the beginning of this section is displayed.
- (8) *show on "def all names up*  
 asks for all names used in a definition; i.e.,  $\beta \gamma \delta \epsilon \zeta \theta$   
 $\mu \nu$
- (9) *show "def names up 1*  
 asks for all names used in a definition and the names of those definitions; i.e.,
- |                  |                          |                              |
|------------------|--------------------------|------------------------------|
| $\beta(\alpha)$  | $\epsilon(\beta \gamma)$ | $\mu(\epsilon \zeta \theta)$ |
| $\gamma(\alpha)$ | $\zeta(\gamma)$          | $\nu(\zeta \theta)$          |
| $\delta(\alpha)$ | $\theta(\delta)$         |                              |
- (10) *show "def names up 2*  
 displays the trees:
- |                                |                                                                |                                                                    |
|--------------------------------|----------------------------------------------------------------|--------------------------------------------------------------------|
| $\frac{\alpha}{\beta(\alpha)}$ | $\frac{\delta(\alpha)}{\epsilon(\beta(\alpha)\gamma(\alpha))}$ | $\frac{\theta(\delta(\alpha))}{\mu(\zeta(\gamma) \theta(\delta))}$ |
| $\gamma(\alpha)$               | $\zeta(\gamma(\alpha))$                                        | $\nu(\zeta(\gamma) \theta(\delta))$                                |
- $\alpha$  is marked for not being in any definition; that is, it is in some sense, independent.
- (11) *show "def words up*  
 asks for the words, other than names, used in every definition; viz.,  $\beta \eta \iota \kappa \lambda \epsilon \pi$
- (12) *show "def '\$\beta\$' def up 1*  
 asks for the definitions defined directly in terms of  $\$ \beta$ ;  
 viz.,  $\alpha \equiv \$ \beta$  or  $\$ \gamma$  or  $\$ \delta$

- (13) *show "def β def up 1*  
 asks for all definitions defined directly in terms of the  
 word β; namely,  $\beta \equiv \beta$  or  $\beta \epsilon$

#### MOVEMENT ALONG THE SCROLL

Data called up by the *show* command will generally not fit within the confines of the data area of the display terminal screen. In such cases, as much as can fit is displayed there; and the examiner must issue further commands to browse through the "off-screen" data.

The product of a *show* can be considered as a continuous sequence of text, akin to a scroll, and the display screen of the terminal as a movable window. The examiner can effectively move the window to a particular point on the scroll (an absolute move) or he can move the window so many units in either direction from its current position (a relative move). He can specify, in an absolute move, a particular name or word in "def", or in a patent file, the number of the patent, the name of the patent section, and the number of the paragraph therein. In a relative move, he can specify how many names, definitions, words, patents, patent sections, or paragraphs the window should be displaced, or alternatively, how many "window's-worth" (frames) should be skipped. He also has the option, in a relative move, of jumping to the next or previous paragraph or frame containing a PWORD from the mark argument to the prior *show* command, or skipping to the next or previous boldface number (used to jump to portions of the text referring to part numbers in the patent drawings). Defaults are set up to facilitate rapid forward scanning of material: entering the window-moving command with no arguments causes the next frame to be displayed.

The command to move the display window through a file has the following form:

MOVE ABSOLUTE RELATIVE

where: MOVE  $\rightarrow \begin{Bmatrix} \text{move} \\ m \end{Bmatrix}$

ABSOLUTE  $\rightarrow \begin{Bmatrix} \text{to} \\ \wedge \end{Bmatrix} \left\{ \begin{array}{c} \text{PWORD} \\ \text{PATENT INTEGER} \\ \wedge \end{array} \right\} \left\{ \begin{array}{c} \text{SECTIONNAME} \\ \wedge \end{array} \right\} \left\{ \begin{array}{c} \text{PARAGRAPH INTEGER} \\ \wedge \end{array} \right\}$

RELATIVE  $\rightarrow$   $\left\{ \begin{array}{c} \text{DOWN} \\ \wedge \\ \text{UP} \end{array} \right\}$  NUM  $\left\{ \begin{array}{l} \text{DEFINITION} \\ \text{NAME} \\ \text{WORD} \\ \text{PATENT} \\ \text{PATENTSECTION} \\ \text{PARAGRAPH} \\ \text{FRAME} \\ \wedge \\ \text{MARK} \\ \# \text{ INTEGER} \end{array} \right\}$

PATENT  $\rightarrow$   $\left\{ \begin{array}{l} \text{patents} \\ \text{patent} \\ \text{pat} \end{array} \right\}$

PATENTSECTION  $\rightarrow$   $\left\{ \begin{array}{l} \text{patentsections} \\ \text{patentsection} \\ \text{ps} \end{array} \right\}$

PARAGRAPH  $\rightarrow$   $\left\{ \begin{array}{l} \text{paragraphs} \\ \text{paragraph} \\ \text{par} \\ \text{p} \end{array} \right\}$

DOWN  $\rightarrow$   $\left\{ \begin{array}{l} \text{down} \\ d \end{array} \right\}$

UP  $\rightarrow$   $\left\{ \begin{array}{l} \text{up} \\ u \end{array} \right\}$

NUM  $\rightarrow$   $\left\{ \begin{array}{l} 1 \\ \wedge \\ 2 \\ 3 \\ \text{[and so on]} \\ \text{last} \end{array} \right\}$

FRAME  $\rightarrow$   $\left\{ \begin{array}{l} \text{frames} \\ \text{frame} \\ f \end{array} \right\}$

MARK  $\rightarrow$   $\begin{Bmatrix} \text{marks} \\ \text{mark} \\ \text{hits} \\ \text{hit} \end{Bmatrix}$

where PWORD, SECTIONNAME, and INTEGER are defined as in rules 29, 28, and 32, respectively, of the formal grammar of the search prescription. The terminal symbol # denotes that the integer which follows must be in boldface in the patent text (see example 6 below).

The above notation cannot easily express the interdependent default conditions. Under RELATIVE, the default of *down frame* does not apply if any ABSOLUTE argument is specified. Under ABSOLUTE, the table below shows what default values are assumed if certain ABSOLUTE arguments are specified (denoted by the symbol  $\checkmark$ ) and there are left unspecified (denoted by a blank). The value "same" means whatever patent or patent section presently shown at the top of the data area; the value "first" means the first patent section in each extract of a patent put on the scroll by the previous *show* command.

| Argument     |              |              | Default Values |              |              |
|--------------|--------------|--------------|----------------|--------------|--------------|
| patent       | section name | paragraph    | patent         | section name | paragraph    |
|              |              | $\checkmark$ | same           | same         | $\checkmark$ |
|              | $\checkmark$ |              | same           | $\checkmark$ | 1            |
|              | $\checkmark$ | $\checkmark$ | same           | $\checkmark$ | $\checkmark$ |
| $\checkmark$ |              |              | $\checkmark$   | first        | 1            |
| $\checkmark$ |              | $\checkmark$ | $\checkmark$   | first        | $\checkmark$ |
| $\checkmark$ | $\checkmark$ |              | $\checkmark$   | $\checkmark$ | 1            |
| $\checkmark$ | $\checkmark$ | $\checkmark$ | $\checkmark$   | $\checkmark$ | $\checkmark$ |

Examples of the *move* command should be self-explanatory. Again, pairs are equivalent:

- (1) *move to patent 1234567 claims*  
*m pat 1234567 cl*
- (2) *move to patent 1234567 last paragraph*
- (3) *move down 2 patentsections*  
*m 2 ps*
- (4) *move down 1 frame*  
*m*
- (5) *move down mark*
- (6) *move up #32*      i.e. to boldface 32
- (7) *move d 3 names*
- (8) *move to '\$β'*

It should be noted that all relative movements are relative to the top line of the data area. That is, if the screen shows the last portion of paragraph 3, all of paragraph 4, and finally, the first paragraph of the subsequent patent section, then *move p* brings paragraph 4 to the top line of the data area and *move ps* brings paragraph 1 from the bottom of the screen to the top of the data area. Furthermore, while relative downward movements which cross higher level boundaries are permitted (*move 2 p* is equivalent to *move ps*, in the example from the previous discussion), relative upward movements which would cross higher level boundaries stop at the boundary (e.g., one cannot step upwards paragraph by paragraph (*move up p*) into another patent).

Occasionally the user would like to see what he had just seen prior to the latest *move* command. The command *backup* will carry out this action. Note that two *backup*'s in a row bring the window back to where it was before--*backup* goes back one step in time to the prior frame displayed.

#### NORMAL AND ERROR RESPONSES

The normal response to the *show* and *move* commands is a confirmatory message in the response area, followed by the display of the requested material in the data area. Co-displayed in the data area

will be identification tags: patent number, sectionname, and paragraph number for search product files; name or word for definition product files.

Associated with all files in the system is a set of attributes. Thus if an examiner attempts to *show* the *claims* of "*def*", an appropriate error message will be placed in the response area.

If a movement is requested to a non-existent paragraph, say, or to a level not placed on the scroll by the previous *show* command, an explanatory error message will be displayed in the response area. If an attempt is made to cross a higher boundary in the upwards direction or to run off the scroll, an error message is displayed; but the window is moved to the boundary or to the edge of the scroll.

When undefined names or independent names (see example 10) are discovered, they are listed alphanumerically in the response area.

#### NATURE OF THE DISPLAY PRODUCT

Most of the characteristics of displayed data has already been discussed. To this we can only add that search products will be displayed ordered numerically on the patent number, and definition products will be displayed ordered alphabetically on the PWORDS, unless the order argument of *show* specifies otherwise.

## CHAPTER I

### MISCELLANEOUS CAPABILITIES

The capabilities touched on in this chapter were felt to lie outside the realm of query language design and more in the area of providing a habitable computer environment in which to utilize the language. The distinction is not a sharp one because of the interplay between the two areas near their common boundary. For this reason, we have decided to present these capabilities, although because they are, except for *askuser*, standard fare in the general milieu of interactive systems, our discussions will be brief.

#### MULTIPLE COMMANDS

More than one command can be issued at one time. The exclamation point is used to separate commands. For example

*find pears|prunes/s/ab! show patents marked pitted*

An exclamation point terminating the last command in a sequence (here, . . . *pitted!*) is unnecessary, but acceptable.

#### PROMPTING

Prompting occurs in three cases. One, if a multi-argument command is incompletely specified, prompting will occur for those arguments which remain missing after default conditions have applied. Two, in a training situation, the command *help* followed by any other command name initiates a sequence of questions put to the user pertinent to using that command name. If *help* is entered without an argument, a listing of every command name and its function is displayed in the data area. Three, the user may wish to parameterize certain commands with the definition capability and have the system prompt for these parameters whenever the command is subsequently entered.

To illustrate the use of *help*, a sample protocol in the prompting mode follows:

INPUT:           help find

RESPONSE:       What is the name of the file you would like to  
                  search? Use "file for the file selected at  
                  log-on, "result for search results.

INPUT: "file

RESPONSE: What logical combination of phrases would you like found?  
 (Optional) in sentence or paragraph?  
 (Optional) in which patent sections?  
 End with semi-colon if you want to conjoin additional phrases.

INPUT: pears or prunes in sentence in abstract;

RESPONSE: What other combination of phrases must be present?  
 (Optional) in sentence or paragraph?  
 (Optional) in which patent sections.  
 End with semi-colon to conjoin additional phrases.

INPUT: pitted in claims

RESPONSE: The command find (pears or prunes in sentence in in abstract) and pitted in claims is being executed

For reasons of simplicity, the prompting mode may not reflect the full range of capabilities provided by the system. As seen in the example given here, the search request is restricted to conjunctive normal form.

Certain procedures may be applicable in a variety of situations, and the examiner may wish to capitalize on this uniformity by changing only a parameter for each situation. Suppose, for example, the examiner discovers a search request which has proven meritorious in a number of cases, provided the nature of, say, the mechanical movement is specified in each case. That is, there is a set of search prescriptions, all identical except as shown below:

Rx1: ----- rotary -----

Rx2: ----- sliding -----

Rx3: ----- vibrating -----

One way to assemble these together would be to define a new search prescription:

```
let Rx be ----- $motion -----
```

where *\$motion* is not defined. Then when the command

```
find $Rx
```

is given, an error comment will be flashed noting that *motion* lacks a definition. The examiner could respond with the command sequence

```
let $motion be sliding! find $Rx
```

and accomplish his goal.

The *askuser* capability provides a more sophisticated way of carrying out this same task without creating a dummy definition. Whenever *askuser* [MESSAGE] occurs, in any command in the system, the MESSAGE is displayed in the response area, the expanded command up to that point is displayed in the data area, and the system pauses. The next input from the user is substituted for *askuser* [MESSAGE] in the expansion, and execution of the command proceeds. Returning to the previous example, suppose the examiner had defined *Rx* as

```
let Rx be ----- askuser [Please supply missing value] -----
```

replacing *\$motion* with the *askuser* component. Then when the command *find Rx* is given, the user is prompted for the missing value. He then types in *sliding*, and the command proceeds into execution. Note that *askuser* is a reserved word for all commands, so any other use (highly unlikely!) of that word must be cast in the literal mold.

#### RESTORING THE DISPLAY SCREEN

The command *restore* will redisplay specified portions of the screen which have been erased or garbled in transmission. It has the syntax

```
restore ( status
         [ screen ]
         [ display ]
         input
         response
         [ data ]
         [ frame ] )
```

## LIBRARIES

We anticipate that several kinds of definition files will be available to the examiner: (1) "official" synonyms (in the broadest sense of the word) collected and codified by a separate office within the Patent Office, (2) "unofficial" synonyms coined by the Examiner corps of use to all, (3) more-or-less permanent definitions personal to the examiner, and (4) temporary definitions coined only for use during a particular computer session.

During the log-off procedure, the system will display all \$-prefixed words, in alphanumerical order, which the examiner had newly defined; and it will ask him to indicate which ones he wants to keep. These will be added to his permanent personal collection.

## INTERRUPT

When the examiner transmits the reserved word @@ to the system, either alone or embedded within a stream of characters, the system will cease execution of the current command. The user can then enter a new command, or he can resume execution by entering the command *resume*.

# APPENDIX I

## PATENT SECTION NAMES

The Patent Office has divided a patent, as stored in the computer, into 32 patent sections. The names and abbreviated names which we has assigned these sections are presented in the table below. This table should be considered the completion of rule 32 in the section, FORMAL GRAMMAR OF THE SEARCH PRESCRIPTION, in Chapter F.

| <u>Contents</u>                      | <u>Name</u>                   | <u>Abbreviated Form</u> |
|--------------------------------------|-------------------------------|-------------------------|
| 1. control number                    | <i>controlnumber</i>          | <i>cn</i>               |
| 2. patent number                     | <i>patentnumber</i>           | <i>pn</i>               |
| 3. patent data                       | <i>patentdate</i>             | <i>pd</i>               |
| 4. title                             | <i>title</i>                  | <i>ti</i>               |
| 5. author                            | <i>author</i>                 | <i>au</i>               |
| 6. assignee                          | <i>assignee</i>               | <i>as</i>               |
| 7. notice data                       | <i>notice</i>                 | <i>no</i>               |
| 8. application number                | <i>appnumber</i>              | <i>an</i>               |
| 9. date of application               | <i>appdate</i>                | <i>ad</i>               |
| 10. international priority           | <i>intlpriority</i>           | <i>ip</i>               |
| 11. related application data         | <i>relatedappdata</i>         | <i>rad</i>              |
| 12. original U.S. classification     | <i>usclass</i>                | <i>usc</i>              |
| 13. official U.S. cross references   | <i>officialxref</i>           | <i>oxref</i>            |
| 14. unofficial U.S. cross references | <i>unofficialxref</i>         | <i>uxref</i>            |
| 15. examiner's digest                | <i>digest</i>                 | <i>di or exd</i>        |
| 16. international classification     | <i>intlclass</i>              | <i>ic</i>               |
| 17. international cross references   | <i>intlxref</i>               | <i>ixref</i>            |
| 18. field of search                  | <i>fieldsofsearch</i>         | <i>fs</i>               |
| 19. patent references cited          | <i>patentref</i>              | <i>pref</i>             |
| 20. other references cited           | <i>otherref</i>               | <i>oref</i>             |
| 21. assistant examiner's name        | <i>asstexaminer</i>           | <i>aex</i>              |
| 22. primary examiner's name          | <i>examiner</i>               | <i>ex</i>               |
| 23. attorney's name                  | <i>attorney</i>               | <i>at</i>               |
| 24. number of claims allowed         | <i>numberofclaims</i>         | <i>ncl</i>              |
| 25. number of drawing figures        | <i>numberofdrawingfigures</i> | <i>ndf</i>              |
| 26. number of drawing sheets         | <i>numberofdrawingsheets</i>  | <i>nds</i>              |
| 27. abstract                         | <i>abstract</i>               | <i>ab</i>               |
| 28. division information             | <i>division</i>               | <i>div</i>              |
| 29. brief summary                    | <i>briefsummary</i>           | <i>bs</i>               |
| 30. drawing description              | <i>drawingdescription</i>     | <i>drd</i>              |
| 31. detailed description             | <i>detaileddescription</i>    | <i>ded</i>              |
| 32. claims                           | <i>claims</i>                 | <i>cl</i>               |

*text* = *ti+ab+bs+drd+ded+cl*

*patent* = all 32 patent sections

## APPENDIX II

### THE PROTOTYPE SYSTEM

#### OVERVIEW OF CAPABILITIES

A working prototype model of the design system was built to demonstrate the feasibility of the concept. It is programmed in PL/I, and it runs under TSO on the IBM 360/370. The prototype was installed at the Patent Office during the contract period, in a configuration of two Computer Optics CO-75 display terminals remotely connected over dual 2400 baud lines to an IBM 360/91 at the Johns Hopkins Applied Physics Laboratory in Silver Spring, Maryland.

The searching capability is basically the same as presented in Chapter F of this report, without NEGATION, PERMUTATION, REPETITION, INTERVAL, WORD INTERNAL, sentence SCOPE, and STEM with bounded span. The domain of the search procedure is limited to the *controlnumber*, *patentnumber*, and *text* patent sections. In addition, the level argument of *show* is augmented and placed in the *find* command instead, thereby allowing the size of *"result* to be reduced in certain cases.

The defining capability is realized through the *let* command only.

The browsing capability is sufficiently different that a new command name, *extract*, was chosen. *Extract* puts the system in a special mode which allows the window to be moved absolutely and relatively without entering the command name *move* but just its arguments. The command *exit* or *end* or *quit* must be entered, then, to get out of this mode. In addition, the domain of browsing is limited to *"result* (because of its different nature, it is termed *"list* in the initial implementation).

None of the miscellaneous capabilities mentioned in Chapter I are implemented.

## SEARCHING

The command to initiate a search is of the form:

*find*  $\left\{ \begin{array}{l} \text{SOURCE} \\ \text{DEPTH} \end{array} \right\} \left\{ \begin{array}{l} \text{DEPTH} \\ \text{SOURCE} \end{array} \right\} \left\{ \begin{array}{l} \text{with} \\ \wedge \end{array} \right\} q$

### Non-terminal Rules

$q \rightarrow \left\{ \begin{array}{l} x \text{ IN } s \\ r \end{array} \right\}$

$r \rightarrow \left\{ \begin{array}{l} r' \\ r \text{ AND } r \end{array} \right\}$

$r' \rightarrow \left\{ \begin{array}{l} (r) \\ r' \text{ OR } r' \\ p \end{array} \right\}$

$p \rightarrow \left\{ \begin{array}{l} (x \text{ IN } s) \\ x \end{array} \right\}$

$x \rightarrow \left\{ \begin{array}{l} x' \\ x \text{ AND } x \\ g \end{array} \right\}$

$x' \rightarrow \left\{ \begin{array}{l} (x) \\ x' \text{ OR } x' \end{array} \right\}$

$g \rightarrow \left\{ \begin{array}{l} g' \\ g \text{ CONC } g \\ g \text{ CONC } k \text{ CONC } g \end{array} \right\}$

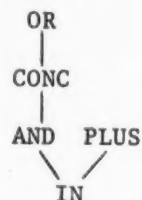
$g' \rightarrow \left\{ \begin{array}{l} (g) \\ g' \text{ OR } g' \\ \text{WORD} \end{array} \right\}$

$k \rightarrow \left\{ \begin{array}{l} j \\ k \text{ CONC } k \end{array} \right\}$

$j \rightarrow \left\{ \begin{array}{l} (k) \\ <g> \\ \text{JOKER} \end{array} \right\}$

$s \rightarrow \left\{ \begin{array}{l} \text{PARAGRAPH} \\ \text{PARAGRAPH IN } n \\ n \end{array} \right\}$

### Precedence:



$n \rightarrow \left\{ \begin{array}{l} (n) \\ n \text{ PLUS } n \\ \text{SECTIONNAME} \end{array} \right\}$

Terminal Classes:

SOURCE  $\rightarrow \begin{Bmatrix} on \\ \wedge \end{Bmatrix} \begin{Bmatrix} "file" \\ \wedge \\ "list" \end{Bmatrix}$

DEPTH  $\rightarrow \begin{Bmatrix} all \\ \wedge \end{Bmatrix} \begin{Bmatrix} PATENT \\ \wedge \\ PATENTSECTION \\ PARAGRAPH \end{Bmatrix}$

PATENT  $\rightarrow \begin{Bmatrix} patents \\ patent \\ pat \end{Bmatrix}$

PATENTSECTION  $\rightarrow \begin{Bmatrix} patentsections \\ patentsection \\ ps \end{Bmatrix}$

PARAGRAPH  $\rightarrow \begin{Bmatrix} paragraphs \\ paragraph \\ par \\ p \end{Bmatrix}$

OR  $\rightarrow \begin{Bmatrix} or \\ | \end{Bmatrix}$

AND  $\rightarrow \begin{Bmatrix} and \\ \& \end{Bmatrix}$

IN  $\rightarrow \begin{Bmatrix} in \\ / \end{Bmatrix}$

PLUS  $\rightarrow \begin{Bmatrix} plus \\ + \end{Bmatrix}$

SECTIONNAME → {

|  |   |                      |   |
|--|---|----------------------|---|
|  | { | titles               | } |
|  |   | title                | } |
|  |   | ti                   | } |
|  |   | abstracts            | } |
|  |   | abstract             | } |
|  |   | ab                   | } |
|  |   | briefsummaries       | } |
|  |   | briefsummary         | } |
|  |   | bs                   | } |
|  |   | drawingdescriptions  | } |
|  |   | drawingdescription   | } |
|  |   | drd                  | } |
|  |   | detaileddescriptions | } |
|  |   | detaileddescription  | } |
|  |   | ded                  | } |
|  |   | claims               | } |
|  |   | cl                   | } |
|  |   | patentnumber         | } |
|  |   | pn                   | } |
|  |   | controlnumber        | } |
|  |   | cn                   | } |

}

JOKER → {

|  |   |         |       |
|--|---|---------|-------|
|  | { | then    | }     |
|  |   | *       | }     |
|  |   | INTEGER | {     |
|  |   |         | {     |
|  |   |         | words |
|  |   |         | word  |
|  |   |         | Λ     |

}

WORD → {

|  |   |                                            |   |
|--|---|--------------------------------------------|---|
|  | { | .                                          | } |
|  |   | [any word as defined by the Patent Office] | } |

INTEGER → {

|  |   |             |   |
|--|---|-------------|---|
|  | { | 1           | } |
|  |   | 2           | } |
|  |   | 3           | } |
|  |   | 4           | } |
|  |   | 5           | } |
|  |   | [and so on] | } |

The actual search procedure is implemented through an index of word types with pointers to the token occurrences of these words in the text. The Patent Office supplied a list of forbidden words, which were not to be in the index because of their insignificance. If the examiner mistakenly specifies a forbidden word in his search prescription, the single joker symbol is substituted and a warning message is flashed in the response area; e.g., *welded the sprocket* becomes *wel~~d~~ed . sprocket*.

The result of a search is, from the user's point of view, a set of texts which match the search prescription. This set is available to the examiner under the name "*list*". The depth argument (PARAGRAPH, PATENTSECTION, or PATENT) specifies the level of text directly accessible on "*list*". If the level specified is narrower than the scope of the search prescription, then the scope is changed to match the depth level when the search is performed. For example:

|                                              |                                                                                                   |
|----------------------------------------------|---------------------------------------------------------------------------------------------------|
| <i>find pears and plums</i>                  | delivers patents in which both<br><i>pears</i> and <i>plums</i> occur                             |
| <i>find pears and plums in<br/>par</i>       | delivers patents in which both<br><i>pears</i> and <i>plums</i> occur in the<br>same paragraph    |
| <i>find all par with pears<br/>and plums</i> | delivers paragraphs in which<br>both <i>pears</i> and <i>plums</i> occur in<br>the same paragraph |

#### DISPLAYING

There are three types of actions available through the display commands:

1. Specification of which portions of text from documents selected by the immediately prior search should be displayed;
2. Control of what parts of the text are displayed on the screen of the display terminal;
3. Termination of the sequence of display commands.

Each of these actions, and the associated commands, are explained in detail below.

Execution of a *find* command can produce a set of "hits", consisting of text portions which satisfied the pattern specified. These results of a search can be considered as a continuous sequence

of text called a hit scroll. The *extract* command (described below) permits a user to designate the hit scroll as the text which he wishes to display, or to specify by name those sections of the hit documents which he wishes to display instead of, or in addition to, the hit sections and paragraphs. This specification defines the display scroll. The choice of "scroll" to describe the text being displayed indicates the way in which the user may browse through the text. The display screen is considered a window and the text can be rolled up and down so that different portions are visible.

The *extract* command defines the display scroll and causes the first 20 lines to appear on the display screen. This command may be used any time after a *find* command has been executed which produced at least one hit document. Any number of *extract* commands may be given to vary the display scroll for a given set of search results.

The syntax of the *extract* command is:

*extract* n'

where:

$$n' \rightarrow \left\{ \begin{array}{l} n' \text{ CONC } n' \\ \text{SECTIONNAME} \\ | \text{hits} | \\ \wedge \end{array} \right\}$$

Note that any section, whether searchable or not, may be displayed. Named patent sections are merged with the hit scroll in the order shown to form the display scroll.

The text shown on the display by *extract* is called a frame. Two pointers define the contents of the frame; each is a vector whose value is a document, section, paragraph, and line number from the display scroll. The TOP pointer has the value of the first line in the frame, while the BOTTOM pointer corresponds to the last line in the frame. After an *extract* command, TOP will point to the first line of the first paragraph of the first section of the first document in the display scroll.

In order to allow browsing through the remaining text, display control commands provide the means for changing the values of TOP and BOTTOM and then generating a new display frame.

To view the next 20 lines on the display scroll, the examiner issues the command

$$\begin{array}{|c|} \hline \text{next} \\ \hline \Lambda \\ \hline \end{array} \quad \begin{array}{|c|} \hline \text{frame} \\ \hline f \\ \hline \end{array}$$

This command sets TOP to the line following the value of BOTTOM, which is equivalent to rolling the scroll to the next sequence of text with no overlap between the new frame and the previous frame. If the next line is in a new paragraph, section, or document, these values in TOP will be set accordingly.

When a new frame appears on the display and the user wishes to review the frame it replaced, he issues the command

$$\begin{array}{|c|} \hline \text{restore} \\ \hline r \\ \hline \end{array} \quad \begin{array}{|c|} \hline \text{frame} \\ \hline f \\ \hline \Lambda \\ \hline \end{array}$$

This command sets the values of TOP and BOTTOM to the values they had in the frame prior (in time) to the one which is being displayed.

To understand the absolute move commands and the remaining relative move commands, it is important to realize that the values of each of the pointers TOP and BOTTOM represent levels in the structure of the text, where "document" is the highest and "line" is the lowest level. A single move command may specify the new value for any or all of the three highest levels of a pointer. These values are specified to the values of BOTTOM when the command is given and then define the new value of TOP. In other words, a move command indicates how much text to skip from what can be seen in the current frame to what will be shown in the next frame.

The syntax of a relative move is:

$$m \rightarrow \begin{array}{|c|} \hline m \text{ CONC } m \\ \hline m' \\ \hline \end{array}$$

where:

$$m' \rightarrow \left\{ \begin{array}{c} \text{DOWN} \\ \Lambda \\ \text{UP} \\ \text{next} \end{array} \right\} \left\{ \begin{array}{c} 0 \\ 1 \\ \Lambda \\ 2 \\ 3 \\ \text{[and so on]} \end{array} \right\} \left\{ \begin{array}{c} \text{PATENT} \\ \text{PATENTSECTION} \\ \text{PARAGRAPH} \\ \Lambda \end{array} \right\}$$

$$\text{DOWN} \rightarrow \left\{ \begin{array}{c} \text{down} \\ d \end{array} \right\}$$

$$\text{UP} \rightarrow \left\{ \begin{array}{c} \text{up} \\ u \end{array} \right\}$$

subject to the additional constraints that (1) a particular level cannot be mentioned more than once and (2) only one level can be defaulted by specifying just direction and quantity (this prevents ambiguity).

DOWN is the default value if no direction is specified, and 1 is the default value if no quantity is specified. *Next* is equivalent to DOWN 1.

Thus, the following commands are all equivalent:

*next par*  
*down par*  
*1 par*  
*par*  
*down 1 par*

The third parameter is the level to be changed. Associated with each command is a default level which is set to the lowest level of move specified in the previous move command. This default level is initially PATENT and is changed by move commands other than those which move frames. The assignment of default levels to a sequence of commands is shown below:

current default level

Patent  
Patent  
Patentsection  
Patentsection  
Par  
Par  
Pat

next command

down  
up ps  
~~frame~~  
1 par 1 ps  
restore  
next pat

An absolute move, unlike a relative move, does not name a direction directly. It can indicate that the TOP pointer should be set to the first or last instance of a level as follows:

$$m \rightarrow \begin{pmatrix} m \text{ CONC } m \\ m'' \end{pmatrix}$$

where:

$$m'' \rightarrow \begin{pmatrix} \text{first} \\ t \end{pmatrix} \begin{pmatrix} \text{PATENT} \\ \text{PATENTSECTION} \\ \text{PARAGRAPH} \\ \Lambda \end{pmatrix}$$

subject to the same two constraints on repetition and defaults which held for relative moves.

The TOP pointer may be set to a particular patent section in the display scroll for a document by the command

SECTIONNAME

The TOP pointer may be set to a particular paragraph within a section by the following syntax:

# INTEGER PARAGRAPH

For example, #3 par signifies the third paragraph.

Certain additional conventions are associated with move commands. A single move command may specify the new value for one to three levels (document, patent section, paragraph) in any order. The actual change in pointer values will be processed in order from the highest to the lowest of the three levels. A change at one level automatically implies that the lower levels are positioned to the first instance in the display scroll associated with the new position of the higher level. For example, if a command specifies a new position at the document level, then the value of the pointer is set to the first section on the display scroll for that document and to the first paragraph in the section which is on the display scroll. If other specifications are given for changes in the pointer values for patent section and/or paragraph levels, these will each be evaluated relative to the position resulting from the higher level moves.

A move at one level may affect the values of the pointer for higher levels also when the move is in the downward direction. This occurs when the amount of the move exceeds the number of instances of that level remaining; e.g., a move *down 3 paragraphs* when there are only two paragraphs left in the current section. In this case, the pointer will automatically move down one at the next higher level in order to obtain a new instance of the lower level. If the lower level is patent and no more patents are on the scroll, then "END OF DATA" will be displayed. For moves upward, the behavior is different. If the move at a level is beyond the number of preceding instances at the level, then the pointer is set to the first instance of that level and the user will be informed that this has occurred. For example, if the pointer is on the second paragraph in a section and the move specified is UP more than one paragraph, the first paragraph will be the new value of the paragraph level.

Following an *extract* command, any number of display movement commands may be given. The status area of the display will indicate

EXTRACT -- EXECUTING.

and will also prompt the user to

ENTER EXTRACT INSTRUCTION

for specifying display movement. In order to leave the display mode, a special command, *exit* or *end* or *quit*, must be given to return to the level of the system where new commands may be given. Prior to giving this instruction, commands such as *extract* and *find* will not be recognized and cause error messages to be displayed.



Figure 3. The computer system: display terminal used to access patents stored in a remote computer

File: Photochemistry      Command status: FIND working      Clock 10:18  
 Message      Session 32 min

find ((photosens' <glass> \$substrate) & deposition within 10 words) & (evaporation masks 2  
 printed circuits) in abstract or claims!

Message understood. Searching has begun.

Expanded search prescription:  
 ((photosens' <glass> (substrate|surface|medium)) & deposition within 10 words) & (evaporation masks  
 2 printed circuits) in abstract or claims

# Word statistics:

|             | Count | Docs |
|-------------|-------|------|
| circuits    | 562   | 132  |
| deposition  | 72    | 65   |
| evaporation | 246   | 134  |
| glass       | 95    | 12   |
| masks       | 75    | 3    |
| medium      | 0     | 0    |
| photosens'  | 2763  | >265 |
| printed     | 33    | 9    |
| substrate   | 98    | 43   |
| surface     | 34    | 15   |

Figure 4. An early experimental form of the screen layout, exemplifying character quality and screen capacity. The division into the four areas of system status, user input, system response, and data has been retained; the query language and the system messages have been revised somewhat in the final version.

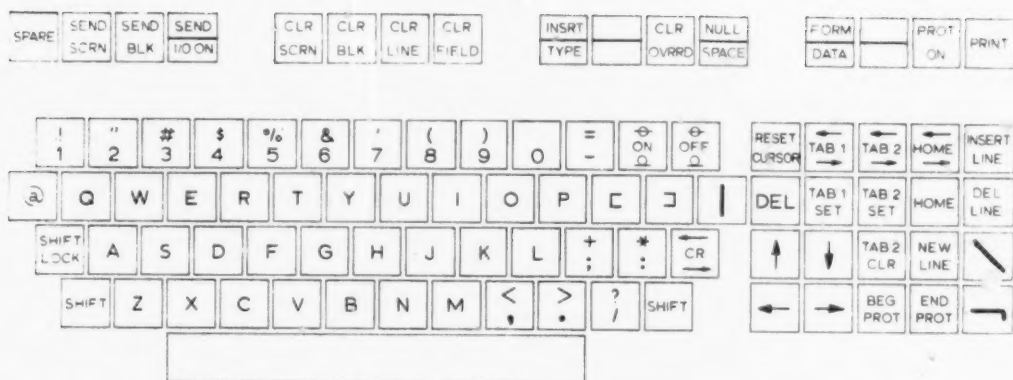


Figure 5. Keyboard engraving (courtesy Computer Optics, Inc.)